Microsoft® C

Reference

Microsoft®

Microsoft® C
REFERENCE

FOR MS® OS/2 and MS-DOS®
OPERATING SYSTEMS

MICROSOFT CORPORATION

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Microsoft.

©Copyright Microsoft Corporation, 1990. All rights reserved.

Printed and bound in the United States of America.

Microsoft, MS, MS-DOS, CodeView, QuickC, and XENIX are registered trademarks and Windows is a trademark of Microsoft Corporation.

IBM is a registered trademark of International Business Machines Corporation.

UNIX is a registered trademark of American Telephone and Telegraph Company.

Document No. LN06515-1189 OEMO711-6Z
10 9 8 7

Contents

Introduction v

PART 1 Utilities

BIND 3
CL (Compiler) 4
CodeView 11
CVPACK 21
EXEHDR 22
EXP 23
HELPMAKE 24
ILINK (Incremental Linker) 27
LIB 28
LINK (Linker) 29
Default File-Name Extensions 33
Module-Definition Files 33
NMAKE 43
NMAKE Macro Statements 44
Pseudotargets 48
Programmer's WorkBench 49
Categories of PWB Functions 50
PWB Functions 55
Return Values of PWB Functions 74
PWB Switches 76
Regular Expressions 88
QuickHelp 92
RM 95
UNDEL 96

PART 2 Language Reference

Keywords 99
Statements 99
Escape Sequences 100
Operators 101
Data Type Sizes 102
Preprocessor Directives 103
Preprocessor Operators 104
Pragmas 105

PART 3 Run-Time Library Functions

Routines by Category 111
Alphabetical Listing 135

Appendixes

A printf/scanf Format Specifiers 369
B Compiler Limits and Numerical Ranges 373
C ASCII Codes 377
D Key Codes 379

Introduction

The Microsoft® *C Reference* contains essential information about the language, run-time library, and utility programs that comprise version 6.0 of the Microsoft C Professional Development System. This book is aimed at the experienced programmer who needs a particular fact—the meaning of an option, the syntax of a pragma, the arguments to a library function. Much of the information is in lists and tables, organized to help you find it at a glance.

The *C Reference* is designed to complement the other Microsoft C documentation, including *Installing and Using the Professional Development System*, *Advanced Programming Techniques*, and the Advisor (on-line help). For example, while *Installing and Using* introduces you to Microsoft C and teaches you how to use the Programmer's WorkBench (PWB), the *C Reference* assumes that you are already familiar with using PWB. Thus, it lists all the functions and keystrokes, both alphabetically and by use, as well as return values and editor switches—but it does not teach you how to use them. You'll find a discussion about how to optimize C programs to best advantage in *Advanced Programming Techniques*. In this *C Reference* you'll find a complete list of all the compiler options, including those for optimization, and a short description of what each one does—but not which is best in a particular situation.

About This Book

The parts of this book and what they contain are discussed below.

Utilities

Part 1 of the *Microsoft C Reference* summarizes the utility programs included with the Microsoft C Professional Development System. The utilities are listed alphabetically. A brief description and the syntax appear at the beginning of each utility's description, followed by command-line options. The description may also include other useful information. For example, the coverage of LINK lists module-definition file statements used in Windows™ and OS/2 applications; the pages on the CodeView® debugger list dialog commands as well as size and format specifiers.

Language Reference

Part 2 includes the basic elements of the C language: it lists keywords and escape sequences, and gives the syntax and a short explanation of statements, preprocessor directives, and pragmas. It also provides tables of information on operators and data type sizes.

Run-Time Library Functions

The Microsoft C libraries contain over 500 functions. Part 3 of this book first lists the functions by category. It then provides an alphabetical overview of each function, listing the function prototype, the include file, the parameter list, a brief description, and compatibility.

Appendixes

The appendixes contain various useful tables, such as one on **printf** and **scanf** formatting codes. They also list compiler limits and program limits at run time, the data ranges defined in LIMITS.H, and the numerical values defined in FLOAT.H. Finally, there are numeric and ASCII conversion tables.

NOTE *The pages that follow use the term "OS/2" to refer to the OS/2 systems—Microsoft Operating System/2 (MS® OS/2) and IBM® OS/2. Similarly, the term "DOS" refers to both the MS-DOS® and IBM Personal Computer DOS operating systems. The name of a specific operating system is used when it is necessary to note features that are unique to the system.*

Document Conventions

This book uses the following document conventions :

Example	Description
STDIO.H	Uppercase letters indicate file names, segment names, registers, and terms used at the operating-system command level.
_far	Boldface letters indicate C keywords, operators, language-specific characters, and library routines. Within discussions of syntax, bold type indicates that the text must be entered exactly as shown.
expression	Words in italics indicate placeholders for information you must supply, such as a file name. Italics are also occasionally used for emphasis in the text.
[[*option*]]	Items inside double square brackets are optional.

#pragma pack {1|2}	Braces and a vertical bar indicate a choice among two or more items. You must choose one of these items unless double square brackets surround the braces.
`#include <stdio.h>`	This font is used for examples, user input, program output, and error messages in text.
CL *options* [[*files...*]]	Three dots following an item indicate that more items having the same form may appear.

`while()`
`{`
`.`
`.`
`.`
`}`
	A column of three dots tells you that part of the program has been intentionally omitted.

CTRL+ENTER	Small capital letters are used for the names of keys on the keyboard. When you see a plus sign (+) between two key names, you should hold down the first key while pressing the second. The carriage-return key, sometimes marked as a bent arrow on the keyboard, is called ENTER. The cursor-movement keys are called the arrow keys. Individual keys are referred to by their direction (LEFT, UP) or by the name on the key (PGUP).
"argument"	Quotation marks enclose a new term the first time it is defined in text.
`"C string"`	Some C constructs, such as strings, require quotation marks. Quotation marks required by the language have the form " " and ' ' rather than “ ” and ‘ ’.
Color Graphics Adapter (CGA)	The first time an acronym is used, it is often spelled out.

PART 1
Utilities

BIND 3
CL (Compiler) 4
CodeView 11
CVPACK 21
EXEHDR 22
EXP 23
HELPMAKE 24
ILINK (Incremental Linker) 27
LIB 28
LINK (Linker) 29
NMAKE 43
Programmer's WorkBench 49
QuickHelp 92
RM 95
UNDEL 96

Summary

The BIND utility converts protected-mode programs that use Family API functions to access system services, allowing them to run in both real mode and protected mode.

Syntax

BIND *infile* [*implibs*] [*linklibs*] [**/O** *outfile*] [**/N@***file*] [**/N** *functions*] [**/M***mapfile*]

In the syntax above, *infile* contains the name of the OS/2 application, *implibs* contains the name of one or more import libraries, and *linklibs* contains the name of one or more standard libraries and object files.

Options

/HELP	Provides on-line help for BIND. First BIND attempts to execute the QuickHelp program, QH.EXE. If QuickHelp or its database is unavailable, the /HELP option lists BIND syntax and options to the standard output.
/M[AP] *mapfile*	Causes a link map to be generated to *mapfile* for the real mode environment of the executable file.
/N[AMES] *functions*	Allows the listing of functions supported in protected mode only. Use with a list of functions or a file specification preceded by @.
/NOLOGO	Suppresses display of the sign-on banner.
/O[UTFILE] *outfile*	Specifies the name for the bound application, *outfile*.
/?	Displays the syntax of the BIND utility.

Summary The CL utility compiles and links one or more C source files.

Syntax **CL** [[*options*]] [[*filename*]] ... [[*libraries link-options*]]

Options /A {T|S|M|C|L|H} Selects one of these standard memory models:

Option	Comments
/AT	Tiny memory model. Code and data are limited to 64K total. Must link with CRTCOM.LIB. Creates a .COM file for real mode. (Same as /Asnd.)
/AS	Small memory model. Code and data are limited to 64K each. (Same as /Asnd.)
/AM	Medium memory model. Data is limited to 64K. (Same as /Alnd.)
/AC	Compact memory model. Code is limited to 64K. (Same as /Asfd.)
/AL	Large memory model. No limits on code or data. (Same as /Alfd.)
/AH	Huge memory model. Same as large model, but individual arrays can exceed 64K. (Same as /Alhd.)

/Astring Sets up a customized memory model. The *string* consists of three characters in any order, indicating code and data pointer size and segment setup.

Group	Code	Description
Code pointers	s	Small (Near)
	l	Large (Far)
Data pointers	n	Near
	f	Far
	h	Huge
Segment setup	d	SS == DS
	u	SS != DS; DS loaded for each function entry
	w	SS != DS; DS not loaded at function entry

/B1 [*path*]	Invokes an alternative preprocessor called C1L.EXE. Use this option to compile programs that generate the message `the compiler is out of near heap`. The drive and path where C1L.EXE resides are optionally specified in *path*.
/B2 [*path*]	Invokes an alternative compiler pass two called C2L.EXE.
/B3 [*path*]	Invokes an alternative compiler pass three called C3L.EXE.
/C	Preserves comments when preprocessing a file: use only with /E, /P, or /EP.
/c	Compiles without linking. Creates an object file but not an executable file.
/D *id* [= [*value*]]	Defines the symbolic constant *id* to the preprocessor. If *value* is defined, the value of *id* is *value*. If the equal sign is given without *value*, the value of *id* is empty. If *id* is given without the equal sign, the value of *id* is 1.
/E	Preprocesses the source file, copying the result to the standard output and inserting **#line** directives.
/EP	Preprocesses the source file, copying the result to the standard output without **#line** directives.
/F *hexnum*	Sets stack size to *hexnum* bytes (this is the same as /link /STACK:*number*). The value must be expressed in hexadecimal notation.
/Fa [*listfile*]	Produces an assembly listing. If *listfile* is unspecified, /Fa defaults to *sourcefilename*.ASM. Not available with the /qc option.
/Fb*bound-exe*	Creates a bound executable file. Use only with /Lp.
/Fc [*listfile*]	Produces a combined source-assembly code listing. If *listfile* is unspecified, /Fc defaults to *sourcefilename*.COD. Not available with the /qc option.
/Fe *exefile*	Names the executable file.
/Fl [*listfile*]	Generates an object-code listing. If *listfile* is not given, /Fl defaults to *sourcefilename*.COD. Not available with the /qc option.
/Fm [*mapfile*]	Creates a linker map file. If *mapfile* is not given, /Fm defaults to *first-sourcefilename*.MAP.
/Fo *objfile*	Names the object file.

/FPa	Generates floating-point calls and selects the alternate math library. Not available with the /qc option.
/FPc	Generates floating-point calls and selects the emulator library (which uses an 80x87 coprocessor if one is present). Not available with the /qc option.
/FPc87	Generates floating-point calls and selects an 80x87 library (which requires an 80x87 coprocessor at run time). Not available with the /qc option.
/FPi	Generates in-line 80x87 instructions and selects an emulator library (uses an 80x87 coprocessor if one is present). This is the default /FP option.
/FPi87	Generates in-line 80x87 instructions and selects an 80x87 library (which requires an 80x87 coprocessor at run time).
/Fr [*browsefile*]	Generates a standard PWB Source Browser database. If *browsefile* is unspecified, /Fr defaults to *basename*.SBR.
/FR [*browsefile*]	Generates an extended Source Browser database. If *browsefile* is unspecified, /FR defaults to *basename*.SBR.
/Fs [*listfile*]	Produces a source listing. If *listfile* is unspecified, /Fs defaults to *sourcefilename*.LST. Not available with the /qc option.
/Fx[*.xreffile*]	Specifies a name for the Microsoft Macro Assembler (MASM) cross-reference file. If *.xreffile* is unspecified, /Fx defaults to *sourcefilename*.CRF.
/G0	Generates 8086/8088 instructions. This is the default /G option.
/G1	Generates 80186/80188 instructions.
/G2	Generates 80286 instructions.
/Gc	Specifies use of FORTRAN- or Pascal-style function calling and naming conventions.
/Gd	Specifies standard (default) C calling conventions.
/Ge	Enables calls to the stack-checking routine (default).
/Gi	Compiles incrementally (when used in conjunction with the /qc option): only functions that have changed are recompiled. Without /qc, /Gi incrementally links by padding object files. Implies /Li.
/Gm	Stores strings in the constant (CONST) segment. Not available with the /qc option.

/Gr	Enables the new **_fastcall** function to call conventions for eligible functions. When possible, values are passed in registers instead of on the stack.
/Gs	Suppresses generation of calls to the stack-checking routine.
/Gt[*number*]	Places data items greater than or equal to *number* bytes in a new segment. Default is 256 if no number is specified.
/Gw	Generates entry/exit code sequences suitable for use in Microsoft Windows applications.
/GW	Same as /Gw, but generates more efficient entry sequences. Used for code other than user callback functions.
/H *number*	Restricts external names to *number* significant characters. The default is 31 characters. Not available with the /qc option.
/HELP	Calls the QuickHelp utility. If the QuickHelp program is not available, CL displays the most commonly used options to the standard output. This option is not case sensitive.
/I *directory*	Adds *directory* to the beginning of the list of directories to be searched for include files.
/J	Changes the default for **char** type from signed to unsigned.
/Lc	Causes the linker to create a compatibility mode executable file. Same as /Lr.
/Li[*number*]	Invokes the incremental linker ILINK instead of the standard linker LINK. ILINK runs faster than LINK while creating larger executable files. The optional *number* specifies the byte boundary to which the linker pads all near functions.
/Lp	Causes the linker to create a protected-mode executable file.
/Lr	Causes the linker to create a real-mode executable file.
/link *link-libinfo*	Passes linker options or library names in *link-libinfo* to LINK.
/MA*MASM_option*	Passes the specified option to the Microsoft Macro Assembler (MASM). MASM is automatically invoked for files listed on the command line with the extension .ASM.
/MD	Creates a dynamically linked C run-time library (OS/2 only). Equivalent to /ALw /FPi /G2 /DDLL /DMT. No library search record.
/ML	Statically links the C run-time library as part of a dynamic-link library (OS/2 only). Equivalent to /ALw /FPa /G2 /DMT. Library search record is changed to LLIBCDLL.LIB.

/MT — Enables support for multithread programs (OS/2 only). Equivalent to /ALw /FPi /G2 /DMT. Library search record is changed to LLIBCMT.LIB.

/ND*dataseg* — Sets the data segment name.

/NM*module* — Sets the module name.

/nologo — Suppresses display of the sign-on banner.

/NT*textseg* — Sets the code segment name.

/O[*opt_codes*]. — Controls optimization. When no codes are included, default optimization is enabled. The optional *opt_codes* argument may contain one or more of the following characters:

Code	Description
a	Assumes no aliasing
c	Enables default (block-level) local common expressions
d	Disables all optimizations
e	Enables global register allocation
g	Enables global optimizations and global common expressions
i	Enables generation of intrinsic routines
l	Enables loop optimizations
n	Disables unsafe loop optimizations (default)
p	Improves consistency in floating-point calculations
r	Disables in-line returns from functions
s	Favors smaller code size
t	Favors faster execution speed (default)
w	Assumes no aliases except across function calls (not available with the /qc option)
x	Maximizes optimizations (equivalent to /Oecilgt/Gs)
z	Enables maximum loop and global-register-allocation optimization

/P	Preprocesses the source file and sends output to a file having the base name of the source file and the extension .I (*basename*.I).
/qc	Invokes the quick compile option. The following options produce an error during a quick compile: /Fa, /Fc, /Fl, /FPa, /FPc, /FPc87, /Fs, /Gm, /H, /Ow, /Zc.
/Sl *linewidth*	Sets the line width of source listing in characters per line. Range is 79-132. Default is 79.
/Sp *pagelength*	Sets the page length of source listing in lines per page. Range is 15-255. Default is 63.
/Ss *subtitle*	Specifies *subtitle* for source listing.
/St *title*	Specifies *title* for source listing.
/Ta *asm_srcfile*	Specifies that *asm_srcfile* is to be treated as an assembler source file, whether or not it has an .ASM extension.
/Tc *c-srcfile*	Indicates that *c-srcfile* is a C source file, whether or not it has a .C extension.
/u	Removes (undefines) definitions of all predefined identifiers.
/U *identifier*	Removes the definition of the given predefined identifier.
/V *string*	Copies the version *string* to the object file.
/w	Suppresses compiler warning messages; same as /W0.
/W{0|1|2|3|4}	Sets the output level for compiler warning messages. The default is 1.
/WX	Makes all warnings fatal; no object file is generated when a warning occurs.
/X	Ignores the list of "standard places" in the search for include files.
/Za	Enforces American National Standards Institute (ANSI) language compatibility, disabling extensions specific to Microsoft C.
/Zc	Causes functions declared as **_pascal** to be treated without regard to case. Not available with the /qc option.
/Zd	Generates line-number information required for the SYMDEB debugger.
/Ze	Enables extensions specific to Microsoft C. This is the default /Z option.

/Zg	Generates function prototypes from function definitions and writes declarations to standard output, without compiling the program.
/Zi	Generates symbolic information required by the Microsoft CodeView® window-oriented debugger.
/Zl	Suppresses emission of library search records in the object file.
/Zp[{1|2|4}]	Packs structure members on the specified byte boundary.
/Zr	Generates code that checks for null pointers and out-of-range far pointers (in the CL command, use only with /qc).
/Zs *sourcefiles*	Performs a syntax check only.

Summary

The Microsoft CodeView window-oriented debugger runs the compiled program while simultaneously displaying the program source code, program variables, memory locations, processor registers, and other pertinent information.

Syntax

CV ‖*options*‖ *executablefile* ‖*arguments*‖

To debug protected-mode programs, set IOPL = YES in your CONFIG.SYS file and use the following syntax:

CVP ‖*options*‖ *executablefile* ‖*arguments*‖

Options

/2	Permits the use of two monitors
/25	Starts in 25-line mode
/43	Starts in EGA 43-line mode
/50	Starts in VGA 50-line mode
/B	Starts in black-and-white mode with CGA or EGA
/C*commands*	Executes commands on start up
/D‖*buffersize*‖	Enables disk overlays (DOS only)
/E	Enables Expanded Memory Support (EMS) (DOS only)
/F	Does not swap video pages between CodeView and the program you are debugging; exchanges debug and output screens by flipping between video pages (faster than /S)
/I*number*	Turns nonmaskable interrupts and 8259-interrupt trapping on (/I1) or off (/I0)
/K	Disables installation of keyboard monitors for the program being debugged
/L *dynlib*	Enables CodeView to search OS/2 dynamic-link libraries for symbolic information
/M	Disables CodeView support of the mouse (use this option when debugging an application that supports the mouse)
/N*number*	/N0 tells CodeView to trap; /N1 tells it not to
/O	Enables debugging of multiple processes under OS/2 protected mode
/R	Enables 80386 debug registers (not available under OS/2)

/S	Starts with screen swapping (exchanges screens by changing buffers, primarily for use with graphics programs)
/X	Uses extended memory to increase debugging capacity (DOS only)

CodeView Commands

Action	Keyboard	Mouse
Display help about the selected topic	F1	Click Help menu
Display contents of help	SHIFT+F1	Click Help menu Contents command
Go to next help screen	CTRL+F1	—
Go to previous help topic	SHIFT+CTRL+F1	—
Go to previously viewed help screen	ALT+F1	Click Back button on help screen
Toggle register window	F2	Click View menu Registers command
Toggle source/assembly/mixed modes	F3	Click Options menu Source Window command
Toggle memory window formats	SHIFT+F3	Click Options menu Memory Window command
Switch to output screen	F4	Click View menu Output command
Close window	CTRL+F4	Click button in upper left corner of window
Go to next Breakpoint or to program end	F5	Click Left button on Go on status line
Switch to next window	F6	Click desired window
Switch to previous window	SHIFT+F6	Click desired window
Execute to cursor	F7	Click Right button at location on status line
Trace into procedure	F8	Click Left button on Trace
Display previous command in history	SHIFT+F8	—
Change window size	CTRL+F8	Click Left button on window border and drag
Toggle Breakpoint at line with cursor	F9 at location	Double-click Left button at location and drag
Step over procedure	F10	Click Left button on Step

Action	Keyboard	Mouse
Display next command in history	SHIFT+F10	—
Maximize window	CTRL+F10	Click button in upper right corner of window
Change flag in register window	Any printing character	Double-click Left button on flag
Delete character at cursor	DEL	—
Toggle insert and overstrike modes	INS	—
Copy text into delete buffer	CTRL+INS	Click Edit menu Copy command
Paste text from delete buffer	SHIFT+INS	Click Edit menu Paste command
Move to next command (Command window only)	TAB	Click Left button at location
Move to previous command (Command window only)	SHIFT+TAB	Click Left button at location
Find selected text	CTRL+\	Click Search menu Selected Text command
Repeat last find	ALT+/	Click Search menu Repeat Find command
Add Watch expression	CTRL+W	Click Watch menu Add Watch command
Delete Watch expression	CTRL+U	Click Watch menu Delete Watch command
Open QuickWatch window for a variable	SHIFT + F9	Click Watch window Quick-Watch command
Scroll up one line in window	CTRL+UP	Click Left button on up arrow on scroll bar
Scroll down one line in window	CTRL+DOWN	Click Left button on down arrow on scroll bar
Scroll up one page in window	PGUP	Click Left button above vertical elevator
Scroll down one page in window	PGDN	Click Left button below vertical elevator
Scroll window to the left	CTRL+PGUP	Click Left button on left arrow or to left of horizontal elevator
Scroll window to the right	CTRL+PGDN	Click Left button on right arrow or to right of horizontal elevator
Move cursor to beginning of line	HOME	Click Left button at location

Action	Keyboard	Mouse
Move cursor to end of line	END	Drag elevator to bottom
Scroll to top of file	CTRL+HOME	Drag vertical elevator to top
Scroll to end of file	CTRL+END	Drag elevator to bottom
Move cursor one word	CTRL+LEFT/ CTRL+RIGHT	Click Left button at location
Move cursor one line	UP/DOWN	Click Left button at location

Dialog Commands

Name	Syntax	Description
address	[[[[*segment* | *register*]]:]]*offset* (*type* *) *constant*	Identifies the location of an expression in various commands
Add Watch	**W?** *expression*[[, *format*]]	Displays *expression* or memory range in the Watch window
Assemble	**A** [[*address*]]	Assembles mnemonics starting at *address*
Breakpoint Clear	**BC** { *list* | * }	Clears breakpoints in *list* or in all breakpoints (*)
Breakpoint Disable	**BD** { *list* | * }	Turns off breakpoints in *list* or in all breakpoints (*)
Breakpoint Enable	**BE** { *list* | * }	Enables breakpoints in *list* or in all breakpoints (*)
Breakpoint List	**BL**	Lists breakpoints with the status of each
Breakpoint Set	**BP** [[*address*]] [[=*symbol*[[*range*]]]] | [[?*expression*]] [[, *passcount*]] [[,"*commands*"]]	Breaks execution when *address* is reached
		Breaks execution when the value of expression changes; if address is listed, the expression is evaluated only at that address
		Breaks execution when expression is true; if address is listed, the expression is evaluated only at that address
Comment	*****comment	Displays explanatory text

Name	Syntax	Description
Compare Memory	**C** *range address*	Compares bytes in *range* with bytes beginning at *address*; displays any mismatched pairs
Current Location	.	Displays the current location
Delay	:	Delays execution of redirected commands (may be repeated for longer delays)
Delete Watch	**Y** {*number* | *}	Deletes (yanks) Watch statements
Display Expression	**?** *expression*[[,*format*]]	Displays *expression* in *format*
Dump	**D**[[*type*]] [[*address* | *range*]]	Dumps memory address or range in *type* format
Enter	**E**[[*type*]] *address* [[*list*]]	Enters memory value in *type* format
Examine Symbols	**X**[[**L***|?|[[*module*!]] |[[*function*.]] [[*symbol*]] [[*]]]]	Displays specified symbols
Execute	**E**	Executes in slow motion
Fill Memory	**F** *range list*	Fills addresses in *range* with values in *list*
Go	**G** [[*breakaddress*]]	Executes to *breakaddress* or to end
List Watch	**W**	Lists current Watch statements
Move Memory	**M** *range address*	Copies values in *range* memory block to *address*
Options	**O**[[*option*]]+ |–[[ ]]	Views or sets CodeView options, including bytes coded (B), flip/swap (F), case sensitivity (C), show symbol address (L), symbols (S), or 386 (3)
Pause	"	Interrupts execution of redirected commands and waits for keystroke
Port Input	**I** *port*	Reads and displays byte from *port*
Port Output	**O** *port byte*	Sends *byte* to *port*
Program Step	**P** [[*count*]]	Executes source lines or instructions, stepping over routine, procedure, and interrupt calls; repeats *count* times
Quick Watch	**??** *symbol*	Displays local variables and complete data structures in a dialog box
Quit	**Q**	Exits and returns to DOS

Name	Syntax	Description
Radix	**N** [[*radixnumber*]]	Sets input radix
Redirection	[[ ]] [[**T**]]>[[>]] [[ ]] < ! = [[*devicename*	Redirects input or output to or from *devicename*
Redraw	**@**	Redraws the screen
Register	**R** [[*registername* [[ ]]=[[*expression*]] [[ ]]	Displays registers and flags, or sets new registers and flags
Restart	**L** [[*arguments*]]	Restarts program
Screen Exchange	\ [[*time*]]	Exchanges the CodeView and output screens
Search	/ [[*regularexpression*]]	Searches for a regular expression
Search Memory	**S** *range list*	Searches *range* for values in *list*
Shell Escape	![[*command*]]	Escapes to a new DOS or OS/2 shell and executes *command*
Source Display Mode	**S** [[ ]] + ! – ! & [[ ]]	Sets display mode to source, assembly, or mixed
Stack Trace	**K**	Displays active routines on the stack
Tab Set	#*number*	Sets *number* spaces for each tab character
Trace	**T** [[*count*]]	Executes source lines or instructions, tracing into routine, procedure, or interrupt calls; repeats *count* times
Unassemble	**U** [[*viewaddress*]]	Displays assembly-language instructions
View	**V** [[[[*viewaddress*]].*line number*]]	Displays source lines
8087	**7**	Displays 80x87 registers

Size Specifiers

Use these data types with Dump and Enter dialog commands:

Data Type	Description
None	Default type
A	ASCII (8-bit) characters
B	Byte (8-bit) hexadecimal values
D	Double-word (32-bit) hexadecimal values

I	Signed integer (16-bit) decimal values; equivalent to C **signed int**
L	Long (64-bit) floating-point (real) values; equivalent to C **double**
S	Short (32-bit) floating-point values; equivalent to C **float**
T	10-byte (80-bit) floating-point values; equivalent to C **long double**
U	Unsigned integer (16-bit) decimal values; equivalent to C **unsigned int**
W	Word (16-bit) hexadecimal values

Format Specifiers

Character	Output Format
d	Signed decimal integer
i	Signed decimal integer
u	Unsigned decimal integer
o	Unsigned octal integer
x | X	Hexadecimal integer
f	Signed value in floating-point decimal format with six decimal places
e | E	Signed value in scientific-notation format with up to six decimal places (trailing zeros and decimal point truncated)
g | G	Signed value with floating-point decimal or scientific-notation format, whichever is more compact
c	Single character
s	Characters printed up to the first null character

NOTE *The prefix* **h** *can be used with the integer-type specifiers (***d***,* **o***,* **u***,* **x***, and* **X***) to specify a* **short int***. The prefix* **l** *can be used with the same types to specify a* **long int***.*

Command-Window Commands (Protected Mode)

Syntax	Description
None	Default type
|	Displays process ID number (PID) and session (screen group) ID number
| *processID*	Enables direct debugging of the child process identified

Thread Commands

Syntax ~[[*specifier*][*command*]] ||

In the syntax above, the *specifier* specifies the thread or threads, and *command* determines debugging activity.

The legal values for *specifier* and their effects are listed below.

Specifier	Function
Blank	Displays the status of all threads. If you omit the *specifier* field you cannot enter a *command*. Instead, you enter the tilde (~) by itself.
#	Specifies the last thread that was executed, which is not necessarily the current thread.
*	Specifies all threads.
number	Specifies the indicated thread, where *number* must be a number corresponding to an existing thread. You can determine corresponding numbers for all threads by entering the command ~*, which gives the status of all threads.
.	Specifies the current thread.

The legal values for *command* and their effects are listed below.

Command	Function
Blank	Displays the status of the selected thread (or threads).
BP	Sets the breakpoint for the specified thread or threads.
E	Executes the specified thread in slow motion. The command ~*E is legal only in source mode, where it executes the current thread in slow motion but lets all other unfrozen threads run.
F	Freezes the specified thread (or threads). A frozen thread will not run in the background or in response to the debugger Go command. However, if you use the **E**, **G**, **P**, or **T** variation of the Thread command, the specified thread is temporarily unfrozen while the debugger executes the command.
G	Passes control to the specified thread until it terminates or until a breakpoint is reached. If you give the command ~*G, all threads execute concurrently (except for those that are frozen). If you specify a particular thread, the debugger temporarily freezes all other threads and executes the specified thread.
P	Executes a program step for the specified thread. The command ~*P is legal only in source mode and causes the debugger to step to the next source line while letting all other threads run (except for those that are frozen). You see only the current thread execute in the debugger display.
S	Selects the specified thread as the current thread. Can apply to only one thread at a time. Thus, the command ~*S results in an error message.
T	Traces the specified thread. Identical to **P**, except that **T** traces through function calls and interrupts, whereas **P** does not.
U	Unfreezes the specified thread or threads. Reverses the effect of a freeze.

Effect of Threads on CodeView Commands

Whether or not you use the Thread Command, the existence of threads affects your CodeView debugging session at all times. Particular debugger commands are strongly affected. Each of these commands is discussed below.

Command	Behavior in Multiple-Thread Programs
.	The Current Location command always uses the current value of CS:IP to determine what the current instruction is. Thus, the Current Location command applies to the current thread.
E	When the debugger is in source mode, the Execute command is equivalent to the ~*E command. The current thread is executed in slow motion while all other threads are also running. When the debugger is in mixed or assembly mode, the Execute command is equivalent to the command ~.P, which does not let other threads run concurrently.
BP	The Breakpoint Set command is equivalent to the ~*BP command; the breakpoint applies to all threads.
G	The Go command is equivalent to the ~*G command; control is passed to the operating system, which executes all threads in the program except for those that are frozen.
P	When the debugger is in source mode, the Program Step command is equivalent to the command ~*P, which lets other threads run concurrently. When the debugger is in mixed or assembly mode, the Program Step command is equivalent to the command ~.P, which lets no other threads run.
K	The Stack Trace command displays the stack of the current thread.
T	When the debugger is in source mode, the Trace command is equivalent to the command ~*T, which lets other threads run concurrently. When the debugger is in mixed or assembly mode, the Trace command is equivalent to the command ~.T, which lets no other threads run.

Summary The CVPACK utility reads through an .EXE file and combines multiple-module debugging information into one table at the end of the file. CodeView can then load the file more quickly.

Syntax **CVPACK** [[*options*]] *filename*

Options

/HELP	Causes CVPACK to attempt to call the QuickHelp program
/P	Packs the file to the smallest possible size

Summary The EXEHDR utility displays and modifies the contents of an executable-file header.

Syntax **EXEHDR** [[*options*]] *filename*

Options

/HEAP:*nnnn*	Sets the heap allocation field to *nnnn* for segmented executable files.
/HELP	Calls the QuickHelp utility. If the QuickHelp program is not available, EXEHDR displays the usage message to the standard output.
/MAX:*nnnn*	Sets the maximum allocation field to *nnnn* paragraphs for DOS.
/MIN:*nnnn*	Sets the minimum allocation field to *nnnn* paragraphs for DOS.
/NOLOGO	Directs EXEHDR to suppress the sign-on banner.
/PMTYPE:*type*	Sets the window type for Presentation Manager programs, where *type* is one of **PM** (equivalent to **WINDOWAPI**), **VIO** (equivalent to **WINDOWCOMPAT**), or **NOVIO** (equivalent to **NOTWINDOWCOMPAT**).
/RESETERROR	Resets the error bit in the header of an OS/2 or Windows executable file. It has no effect on DOS executable files.
/STACK:*nnnn*	Sets the stack allocation field to *nnnn* for DOS and segmented-executable files.
/VERBOSE	Provides more information about protected mode (OS/2) executable files and DLLs, including the default flags in the segment table, all run-time relocations, and additional fields from the .EXE header.

Summary

The EXP utility expunges (removes) all files from the DELETED subdirectory. Copies of deleted files are placed in DELETED by PWB (when the **backup** switch is on) and by RM.

Syntax

EXP [[*options*]] [[*directory*]]

If no *directory* is specified, the current directory's DELETED subdirectory is used.

Options

/HELP	Causes EXP to attempt to call the QuickHelp program
/Q	Specifies quiet mode (the deleted file names are not displayed on the screen)
/R	Causes EXP to operate recursively on all subdirectories

Summary

The HELPMAKE utility creates help files and customizes the help files supplied with Microsoft language products. It creates a help database from one or more input files that contain information specially formatted for the help system.

Syntax

HELPMAKE [*options*] {/E[*n*] | /D} *sourcefiles*

You must supply the /HELP, /E (encode), or /D (decode) option.

Options

/A*c*

Specifies *c* as an application-specific control character for the help database, marking a line that contains special information for internal use by the application.

/C

Indicates that the context strings are case sensitive. At run time, all searches for help topics are case sensitive.

/D[*letter*]

Decodes the input file into its component parts. If a destination file is not specified with the /O option, the help file is decoded to **stdout**. HELPMAKE decodes the file in different ways, depending on the letter specified.

Letter	Effect
/D	"Decode." Fully decodes the help database, leaving all cross-references and formatting information intact.
/DS	"Decode split." Splits the concatenated, compressed help database into its components, using their original names. If the database was created without concatenation (the default), HELPMAKE simply copies it to a file with its original name. No decompression occurs.
/DU	"Decode unformatted." Decompresses the database and removes all screen formatting and cross-references. The output can still be used later for input and recompression, but all of the screen formatting and cross-references are lost.

/E[[*n*]]

Creates ("encodes") a help database from a specified text file (or files). The optional *n* indicates the amount of compression to take place. If *n* is omitted, HELPMAKE compresses the file as much as possible, thereby reducing the size of the file by about 50 percent. The more compression requested, the longer HELPMAKE takes to create a database file. The value of *n* is a number in the range 0–15. It is the sum of successive powers of 2 representing these compression techniques:

Value	Technique
0	No compression
1	Run-length compression
2	Keyword compression
4	Extended-keyword compression
8	Huffman compression

Add values to combine compression techniques. For example, use /E3 to get run-length and keyword compression.

/H

Displays a summary of HELPMAKE syntax and then exits.

/HELP

Calls the QuickHelp utility. If the QuickHelp program is not available, HELPMAKE displays the most commonly used HELPMAKE options to the standard output (without encoding or decoding any files).

/K *filename*

Specifies a file containing word-separator characters. This file should consist of a single line of text containing characters that separate words. ASCII characters from 0 to 32 (including the space), and character 127, are always separators. If the /K option is not specified, the following characters are also considered separators:

```
!"#&'()*+-,/:;<=>?@[\]^_`{|}~
```

/L

Locks the generated file so that it cannot be decoded by HELPMAKE at a later time.

/O*outfile*

Specifies *outfile* as the name of the help database. The name *outfile* is optional with the /D option.

/S*n*

Specifies the type of input file, according to the following *n* values:

Option	File Type
/S1	Rich Text Format (RTF)
/S2	QuickHelp Format (default)
/S3	Minimally Formatted ASCII

/T

During encoding, translates dot commands to application-specific commands. During decoding, translates application commands to dot commands.

/V[*n*]

Indicates the "verbosity" of diagnostic and informational output, depending on the value of *n*. If you omit this option or specify only /V, HELPMAKE gives you its most verbose output. The possible values of *n* are listed below:

Option	Effect
/V	Maximum diagnostic output
/V0	No diagnostic output and no banner
/V1	Prints only HELPMAKE banner
/V2	Prints pass names
/V3	Prints contexts on first pass
/V4	Prints contexts on each pass
/V5	Prints any intermediate steps within each pass
/V6	Prints statistics on help file and compression

/W*n*

Indicates the fixed width of the resulting help text in number of characters. The value of *n* can range from 11 to 255. If /W is omitted, the default is 76. When encoding RTF source (/S1), HELPMAKE automatically formats the text to *n*. When encoding QuickHelp (/S2) or minimally formatted ASCII (/S3) files, HELPMAKE truncates lines to *n* characters.

Summary The ILINK utility invokes the Microsoft Incremental Linker.

Syntax **ILINK** [*options*] *projname* [*modulelist*]

Options

/A	Directs ILINK to check for changes in the object files since the last linking process.
/C	Specifies case sensitivity for all public symbol names.
/E "*commands*"	Specifies commands to be executed if incremental linking fails. The option /E "LINK /INC" is the default.
/HELP	Provides on-line help about the incremental linker. First /HELP attempts to execute the QuickHelp program QH.EXE. If QuickHelp or its database is unavailable, /HELP lists ILINK options to the standard output.
/I	Specifies that only an incremental link is to be attempted. If the incremental link fails, a fatal error follows the incremental violation message.
/NOLOGO	Causes ILINK to suppress the sign-on banner.
/V	Verbose mode. Directs ILINK to list all changed modules.
/X	Prevents ILINK from allocating memory from an expanded memory manager if one is present.

Summary The LIB utility helps create, organize, and maintain run-time libraries.

Syntax LIB *inlibrary* [[*options*]] [[*commands*]] [[,[[*listfile*]] [[,[[*outlibrary*]]]]]] [[;]]

Options

/HELP	Provides on-line help for LIB. First /HELP attempts to execute the QuickHelp program QH.EXE. If QuickHelp or its database is unavailable, /HELP lists the LIB options to the standard output.
/I[[GNORECASE]]	Directs LIB to ignore case when comparing symbols (the default). Use to combine a library marked /NOI with an unmarked library for a new unmarked library.
/NOE[[XTDICTIONARY]]	Prevents LIB from creating an extended dictionary.
/NOI[[GNORECASE]]	Directs LIB not to ignore case when comparing symbols.
/NOL[[OGO]]	Causes LIB to suppress the sign-on banner.
/PA[[GESIZE]]:*n*	Specifies the library-page size of a new library, or changes the library-page size of an existing library. The default page size for a new library is 16 bytes.

Commands

+*filename*	Appends an object file or library file to the given library
-*filename*	Deletes a module from the library
-+*filename*	Replaces a module by deleting it from the library and appending to the library an object file with the same name
**filename*	Extracts a module without deleting it from the library and saves the module as an object file with the same name (copies it)
-**filename*	Extracts a module and deletes it from the library after saving it in an object file with the same name (moves it)

NOTE *Place an ampersand (&) at the end of an input line to continue the command codes on the next line.*

Summary The Microsoft Segmented-Executable Linker (LINK) combines object files into a single executable file.

Syntax **LINK** [*options*] *objfiles* [,[*exefile*] [,[*mapfile*] [,[*libraries*] [,[*deffile*] [ ]]]]] [;]

Options

/A — /A[LIGNMENT]:*size*

Directs LINK to align segment data in the executable file along the boundaries specified by *size*, where *size* must be a power of two. Default is 512.

/BA — /BA[TCH]

Suppresses prompts for library or object files not found. LINK generates error or warning messages instead, if appropriate. Also prevents LINK from printing the sign-on banner and echoing input from response files.

/CO — /CO[DEVIEW]

Creates a special-format executable file containing the symbolic data and line-number information needed by the Microsoft CodeView debugger.

/CP — /CP[ARMAXALLOC]:*bytes*

Sets the program's maximum memory allocation to *bytes*.

/DO — /DO[SSEG]

For assembly-language programs only. Forces segments in the executable file to be ordered as follows:

1. Segments with class name ending in CODE
2. All other segments outside DGROUP
3. DGROUP segments, in the following order:
   a. Segments of class BEGDATA
   b. Segments not of class BEGDATA, BSS, or STACK
   c. Segments of class BSS
   d. Segments of class STACK

/DS

/DS[ALLOCATE]

For assembly-language programs only. Directs the linker to load all data starting at the high end of the data segment instead of the low end.

/E

/E[XEPACK]

Packs the executable file during linking by removing repeated series of bytes.

/F

/F[ARCALLTRANSLATION]

Directs the linker to optimize far calls to procedures that lie in the same segment as the caller. Use in conjunction with /PACKCODE. Far-call translation is turned on by default.

/HE

/HE[LP]

Provides on-line help about the linker. First LINK attempts to execute the QuickHelp program QH.EXE. If QuickHelp or its database is unavailable, LINK lists all available options to the standard output.

/HI

/HI[GH]

Places the executable file as high in memory as possible. For real-mode assembly-language programs only.

/INC

/INC[REMENTAL]

Prepares for subsequent incremental linking with ILINK. Incompatible with /E and /TINY.

/INF

/INF[ORMATION]

Causes the linker to display phase of linking and names of object files being linked to the standard output.

/LI

/LI[NENUMBERS]

Includes source-file line numbers and associated addresses in the map file. In addition, an object file with line-number information must be given to LINK. The /Zd option can be used with most Microsoft compilers to include line numbers in the object file. If LINK is given an object file without line-number information, the /LI option has no effect.

/M

/M[AP]

Creates a listing file containing all public (global) symbols defined in the input modules.

/NOD /NOD[[EFAULTLIBRARYSEARCH]] [[:*filename*]]

Causes default libraries to be ignored. If *filename* is specified, LINK searches all libraries except *filename*.

/NOE /NOE[[XTDICTIONARY]]

Prevents the linker from searching the extended dictionary (an internal list of intermodule dependencies). Use this option to suppress error messages about multiple symbol definitions.

/NOF /NOF[[ARCALLTRANSLATION]]

Turns off far-call translation (translation of far calls to near calls where possible). Far-call translation is off by default.

/NOI /NOI[[GNORECASE]]

Causes the linker to distinguish between uppercase and lowercase letters.

/NOL /NOL[[OGO]]

Causes the linker to suppress the sign-on banner.

/NON /NON[[ULLSDOSSEG]]

Directs the linker to arrange segments in the executable file in the same order as they are arranged by the /DOSSEG option, with no additional bytes at the beginning of the **_TEXT** segment (if it is defined).

/NOP /NOP[[ACKCODE]]

Turns code-segment packing off (if code-segment packing has been turned on).

/O /O[[VERLAYINTERRUPT]]:*number*

Allows the user to select an interrupt number other than 63 (the default) for passing control to overlays.

/PACKC /PACKC[[ODE]] [[:*number*]]

Directs the linker to group neighboring code segments together in the medium, large, and huge memory models, where *number* specifies the maximum size of groups formed by /PACKCODE. When used with /FARCALLTRANSLATION, this produces smaller code.

/PACKD

/PACKD[ATA] [:*number*]

Directs the linker to group neighboring data segments together in the compact, large, and huge memory models, where *number* specifies the maximum size of groups formed by /PACKDATA.

/PADC

/PADC[ODE]:*padsize*

Directs the linker to add filler bytes to the end of each code module for subsequent linking with ILINK: *padsize* specifies the number of bytes.

/PADD

/PADD[ATA]:*padsize*

Adds *padsize* bytes to each data segment.

/PAU

/PAU[SE]

Causes the linker to pause in the linking session so that disks can be changed. This option is provided for floppy-disk users.

/PM

/PM[TYPE]:*type*

Sets the window type for Presentation Manager (PM) programs, where *type* is:

Type	Windows Equivalent
PM	**WINDOWAPI** keyword in a module definition (.DEF) file
VIO	**WINDOWCOMPAT**
NOVIO	**NOTWINDOWCOMPAT**

/Q

/Q[UICKLIB]

Produces a Quick library for use with Microsoft QuickC® (early versions only) or Microsoft QuickBASIC.

/SE

/SE[GMENTS]:*number*

Sets the maximum number of segments the program can have, which can be any positive value up to 3,072 bytes. The default is 128.

/ST

/ST[ACK]:*number*

Sets the stack size to *number*, which can be any positive value up to 65,535 bytes. The default for C programs is 0x800 (2,048) bytes.

/T /T[INY]

Causes the linker to produce .COM files. Incompatible with /INCREMENTAL.

/W /W[ARNFIXUP]

Issues a warning for each segment offset previously at the beginning of a group but altered during linking to no longer be at the beginning of a group.

Default File-Name Extensions

File Type	Default Extension
Object	.OBJ
Executable	.EXE (or .COM with /T option)
Map	.MAP
Standard Library	.LIB
Dynamic-Link Library	.DLL (OS/2 and Windows only)
Module Definitions	.DEF (OS/2 and Windows only)

Module-Definition Files

A module-definition file is required for Windows applications and libraries and for dynamic-link libraries (DLLs) that run under OS/2. Such files are optional (but desirable) for all OS/2 applications. Each file contains one or more module statements that describe the module name, attributes of program segments, and number and names of exported and imported functions. Following the list of module statements below is a description of each one, including syntax and fields.

NAME	**SEGMENTS**	**HEAPSIZE**
LIBRARY	**STACKSIZE**	**PROTMODE**
DESCRIPTION	**EXPORTS**	**OLD**
CODE	**IMPORTS**	**REALMODE**
DATA	**STUB**	**EXETYPE**

NAME Statement

Summary The **NAME** statement identifies the executable file as an application and optionally defines the name and type.

Syntax **NAME** [[*app_name*]] [[*app_type*]]

Fields

app_name — Name of the application. Can be any valid file name.

app_type — {**WINDOWAPI** | **WINDOWCOMPAT** | **NOTWINDOWCOMPAT**}

Defines the type of application to be linked in a Windows environment. **WINDOWAPI** specifies a Presentation Manager (PM) application using the API provided by PM and must be executed in the PM environment. This is equivalent to linker option /PMTYPE:PM. **WINDOWCOMPAT** specifies a PM-compatible application that can run inside a PM window or in a separate screen group using the proper subset of OS/2 video, keyboard, and mouse functions supported in PM applications. This is equivalent to linker option /PMTYPE:VIO. **NOTWINDOWCOMPAT** specifies that the application is not compatible with PM and must operate in a separate screen group from PM. This is equivalent to linker option /PMTYPE:NOVIO.

LIBRARY Statement

Summary The **LIBRARY** statement identifies the executable file as a dynamic-link library. It can specify the name of the library or the type of library-module initialization required.

Syntax **LIBRARY** [[*libraryname*]] [[*initialization*]]

Fields

libraryname — Name of the library. Can be any valid file name.

initialization — {**INITGLOBAL** | **INITINSTANCE**}

Determines the type of initialization required. **INITGLOBAL** specifies that the library-initialization routine is called only when the library module is initially loaded into memory. **INITINSTANCE** specifies that the library-initialization routine is called each time a new process gains access to the library. This option is necessary for DLLs that use the C library functions.

DESCRIPTION Statement

Summary The **DESCRIPTION** statement inserts the specified text into the application or library.

Syntax **DESCRIPTION** *'text'*

The *text* is a one-line string enclosed in single quotation marks.

CODE Statement

Summary The **CODE** statement defines the default attributes for code segments within the application or library. The **SEGMENTS** statement can override this default.

Syntax **CODE** [[*load*]] [[*executeonly*]] [[*iopl*]] [[*conforming*]] [[*shared*]] [[*movable*]] [[*discard*]]

Each attribute field can appear one time at most, and order is not significant.

Fields

load {**PRELOAD** | **LOADONCALL**}

Determines when a code segment is loaded. **PRELOAD** specifies that the segment is loaded automatically at the beginning of the program. **LOADONCALL** (default) specifies that the segment is not loaded until accessed.

executeonly {**EXECUTEONLY** | **EXECUTEREAD**}

Determines whether a code segment can be read as well as executed. **EXECUTEONLY** specifies that the segment can only be executed. **EXECUTEREAD** (default) specifies that the segment can be both executed and read.

iopl {**IOPL** | **NOIOPL**}

Determines whether or not a segment has I/O privilege (OS/2 only). **IOPL** specifies that the code segment has I/O privilege. **NOIOPL** (default) specifies that the code segment does not have I/O privilege.

conforming {**CONFORMING** | **NONCONFORMING**}

Determines whether or not a code segment is a 80286 "conforming" segment. **CONFORMING** specifies that the segment is conforming, which means it can be called from either Ring 2 or Ring 3 and it executes at the caller's privilege level. **NONCONFORMING** (default) specifies that the segment is nonconforming.

shared {SHARED | NONSHARED}

Determines whether all instances of the program can share an **EXECUTEREAD** code segment. **SHARED** specifies that one copy of the code segment is loaded and shared among all processes accessing the module (the default for dynamic-link libraries). **NONSHARED** specifies that the segment cannot be shared and must be loaded separately for each process (the default for applications).

movable {MOVABLE | FIXED}

Determines whether a segment can be moved around in memory. For use with real-mode Windows only. **FIXED** is the default.

discard {DISCARDABLE | NONDISCARDABLE}

Determines whether a segment of memory can be discarded to fill a different memory request. If the discarded memory contains a function that is called later, it is reloaded from disk. For use with real-mode Windows only. **NONDISCARDABLE** is the default.

DATA Statement

Summary The **DATA** statement defines the default attributes for the data segments within the application or module.

Syntax **DATA** [[*load*]] [[*readonly*]] [[*instance*]] [[*iopl*]] [[*shared*]] [[*movable*]] [[*discard*]]

Each field attribute can appear one time at most, and order is not significant.

Fields *load* {PRELOAD | LOADONCALL}

Determines when a code segment is loaded. **PRELOAD** specifies that the segment is loaded automatically at the beginning of the program. **LOADONCALL** (default) specifies that the segment is not loaded until accessed.

readonly {READONLY | READWRITE}

Determines access rights to a data segment. **READONLY** specifies that the segment can only be read. **READWRITE** (default) specifies that the segment can be both read and written to.

instance {NONE | SINGLE | MULTIPLE}

Affects the sharing attributes of the automatic data segment represented by the group name DGROUP. **NONE** specifies that no automatic data segment is created. **SINGLE** specifies that a single automatic data segment is shared by all instances of the module (default for dynamic-link libraries). **MULTIPLE** specifies that the automatic data segment is copied for each instance of the module (default for applications).

iopl {IOPL | NOIOPL}

Determines whether or not a segment has I/O privilege (OS/2 only). **IOPL** specifies that the code segment has I/O privilege. **NOIOPL** (default) specifies that the code segment does not have I/O privilege.

shared {SHARED | NONSHARED}

Determines whether all instances of the program can share an **EXECUTEREAD** code segment. **SHARED** specifies that one copy of the code segment is loaded and shared among all processes accessing the module (the default for dynamic-link libraries). **NONSHARED** specifies that the segment cannot be shared and must be loaded separately for each process (the default for applications).

movable {MOVABLE | FIXED}

Determines whether a segment can be moved around in memory. For use with real-mode Windows only. **FIXED** is the default.

discard {DISCARDABLE | NONDISCARDABLE}

Determines whether a segment of memory can be discarded when not needed. If the discarded memory contains data that is accessed later, it is reloaded from disk. For use with real-mode Windows only. **NONDISCARDABLE** is the default.

SEGMENTS Statement

Summary The **SEGMENTS** statement defines the attributes of one or more segments in the application or library on a segment-by-segment basis. The attributes specified by this statement override defaults set in **CODE** and **DATA** statements.

Syntax **SEGMENTS** [[']] *segmentname*[[']] [[**CLASS** '*classname*']] [[*load*]] [[*readonly*]] [[*executeonly*]] [[*iopl*]] [[*conforming*]] [[*shared*]] [[*movable*]] [[*discard*]]

The **SEGMENTS** keyword marks the beginning of the segment definitions. This keyword can be followed by one or more segment definitions, each on a separate line (limited by the number set by the linker's /SEGMENTS option, or 128 if the option is not used). The **CLASS** keyword specifies the class of the segment.

Fields

load {**PRELOAD** | **LOADONCALL**}

Determines when a code segment is loaded. **PRELOAD** specifies that the segment is loaded automatically at the beginning of the program. **LOADONCALL** (default) specifies that the segment is not loaded until accessed.

readonly {**READONLY** | **READWRITE**}

Determines access rights to a data segment. **READONLY** specifies that the segment can only be read. **READWRITE** (default) specifies that the segment can be both read and written to.

executeonly {**EXECUTEONLY** | **EXECUTEREAD**}

Determines whether a code segment can be read as well as executed. **EXECUTEONLY** specifies that the segment can only be executed. **EXECUTEREAD** (default) specifies that the segment can be both executed and read.

iopl {**IOPL** | **NOIOPL**}

Determines whether or not a segment has I/O privilege (OS/2 only). **IOPL** specifies that the code segment has I/O privilege. **NOIOPL** (default) specifies that the code segment does not have I/O privilege.

conforming {**CONFORMING** | **NONCONFORMING**}

Determines whether or not a code segment is a 80286 "conforming" segment. **CONFORMING** specifies that the segment is conforming, which means it can be called from either Ring 2 or Ring 3 and it executes at the caller's privilege level. **NONCONFORMING** (default) specifies that the segment is nonconforming.

shared {**SHARED** | **NONSHARED**}

Determines whether all instances of the program can share an **EXECUTEREAD** code segment. **SHARED** specifies that one copy of the code segment is loaded and shared among all processes accessing the module (the default for dynamic-link libraries). **NONSHARED** specifies that the segment cannot be shared and must be loaded separately for each process (default for applications).

movable {**MOVABLE** | **FIXED**}

Determines whether a segment can be moved around in memory. For use with real-mode Windows only. **FIXED** is the default.

discard {**DISCARDABLE** | **NONDISCARDABLE**}

Determines whether a segment of memory can be discarded to fill a different memory request. If the discarded memory contains a function that is called later, it is reloaded from disk. For use with real-mode Windows only. **NONDISCARDABLE** is the default.

STACKSIZE Statement

Summary The **STACKSIZE** statement specifies the size of the stack segment. (The **STACKSIZE** statement overrides the /STACKSIZE linker option.)

Syntax **STACKSIZE** *number*

The *number* must be an integer; it is considered to be in decimal format by default, but C notation can be used to specify hexadecimal or octal format.

EXPORTS Statement

Summary The **EXPORTS** statement declares the names and attributes of the functions exported to other modules and the functions that run with I/O privilege. The **EXPORTS** statement also declares Windows callback functions such as dialog boxes and timer functions. It is unnecessary to export callback functions within an OS/2 Presentation Manager program.

Syntax

EXPORTS *entryname* [[=*internalname*]] [[*@ord*[[**RESIDENTNAME**]] [[*pwords*]] [[**NODATA**]]

The **EXPORTS** keyword marks the beginning of the export definitions. It can be followed by up to 3,072 export definitions, each on a separate line.

The optional keyword **NODATA** is ignored by OS/2, but is provided for use by real-mode Windows. It means there is no static data in the function.

Fields

entryname	Defines the function name as known to other modules.
internalname	Defines the actual export function name as it appears within the module. Default is the same as *entryname*.
ord	Defines the function's ordinal position within the module-definition table. Can contain the optional keyword **RESIDENTNAME**, which specifies that the function name be kept resident in memory at all times.
pwords	Specifies the total size of the function's parameters in words.

IMPORTS Statement

Summary

The **IMPORTS** statement defines the names of the functions to be imported for the application or library.

Syntax

IMPORTS [[*internalname*=[[*modulename*.*entry*

The **IMPORTS** keyword marks the beginning of the import definitions. This keyword is followed by one or more import definitions, each on a separate line. Note that if you supply an import library, you need not list the individual functions.

Fields

internalname	Specifies the name that the importing module actually uses to call the function. By default, *internalname* is the same as the name given in *entry*.
modulename	Name of the application or library containing the function.
entry	Determines the function to be imported and can be a name or an ordinal value.

STUB Statement

Summary The **STUB** statement adds a DOS executable file to the beginning of the application or library being created. Typically, the stub displays a message and terminates execution.

Syntax **STUB** *'filename'*

The *filename* specifies the DOS executable file to be added. If the linker does not find *filename* in the current directory, it searches in the list of directories specified in the PATH environment variable.

HEAPSIZE Statement

Summary The **HEAPSIZE** statement defines the size of the application's local heap in bytes. This value affects the size of the automatic data segment.

Syntax **HEAPSIZE** {*bytes* | **MAXVAL**}

The *bytes* field is a decimal integer value by default. However, hexadecimal and octal numbers can be entered by using C notation. **MAXVAL** is an optional keyword which can be substituted for *bytes* to set the field parameter. **MAXVAL** sets the heap size to 64K minus the size of DGROUP.

PROTMODE Statement

Summary The optional **PROTMODE** statement specifies that the module runs only in protected mode and not in Windows or dual mode.

Syntax **PROTMODE**

OLD Statement

Summary The **OLD** statement directs the linker to search another dynamic-link module for export ordinals.

Syntax **OLD** *'filename'*

REALMODE Statement

Summary The **REALMODE** statement specifies that the application runs only in real mode.

Syntax **REALMODE**

EXETYPE Statement

Summary The optional **EXETYPE** statement specifies in which operating system the application (or dynamic-link library) is to run.

Syntax **EXETYPE** [[**OS2** | **WINDOWS** | **UNKNOWN**]]

Summary The NMAKE utility automates the process of compiling and linking project files. NMAKE is upwardly compatible with the older MAKE utility if the /MAKE option is used.

Syntax NMAKE [*options*] [*macrodefinitions*] [*targets*]

Options

/A	Executes commands to build all the targets requested even if they are not out-of-date.
/C	Suppresses the NMAKE copyright message and prevents non-fatal error or warning messages from being displayed.
/D	Displays the modification date of each file when the date is checked.
/E	Causes environment variables to override macro definitions within description files.
/F *filename*	Specifies *filename* as the name of the description file to use. If a dash (–) is entered instead of a file name, NMAKE accepts input from the standard input device instead of using a description file. If /F is not specified, NMAKE uses MAKEFILE as the description file.
/HELP	Calls the QuickHelp utility. If the QuickHelp program is not available, NMAKE displays the most commonly used NMAKE options to the standard output.
/I	Ignores exit codes (also called return or error codes) returned by programs called from the NMAKE description file. NMAKE continues executing the rest of the description file despite the errors.
/N	Displays the commands from the description file that NMAKE would execute, but does not execute these commands. This option is useful for checking which targets are out-of-date and for debugging description files.
/NOLOGO	Suppresses the sign-on banner when NMAKE executes.
/P	Prints all macro definitions and target descriptions.
/Q	Returns a zero exit code if the target is up-to-date and a non-zero exit code if it is not. This option is useful when invoking NMAKE from within a batch file.
/R	Ignores inference rules and macros contained in the TOOLS.INI file.
/S	Suppresses display of commands as they are executed.

/T	Changes the modification dates for outdated target files to the current date. The file contents are not modified.
/X *filename*	Sends all error output to *filename*, which can be either a file or a device. If a dash is entered instead of a file name, the error output is sent to the standard output device.
/Z	Internal option for use by the Programmer's WorkBench.
/?	Displays a brief summary of NMAKE syntax and exits to the operating system.

NMAKE Macro Statements

Description Block Format

Syntax

target... : [[*dependent*]] [[;*command*]] [[#*comment*]]
[[*command*]]
[[#*comment*]]
[[#*comment*]] | [[*command*]]
.
.
.

The following symbols are used in description blocks:

Symbol	Meaning
#	Introduces comment field
*, ?	DOS wild-card characters; NMAKE expands them in target names when it reads the description file
^	Introduces any escape character in a description file, including these: # ( ) $ ^ \ { } ! @

Command Modifiers

These characters can be placed in front of a command to modify its effect. The character must be preceded by at least one space.

Character	Action
–	Turns off error checking for the command
@	Prevents NMAKE from displaying the command as it executes
!	Causes the command to be executed for each dependent file if the command uses one of the special macros $? or $**

Macros

Syntax *macroname=macrostring*

The *macroname* can be any combination of alphanumeric characters and the underscore (_) character. The *macrostring* can be any valid string.

Having defined the macro, use the following reference to include it in a dependency line or command:

$(*macroname*)

Use the following syntax to substitute text within a macro:

$(*macroname*:*string1* = *string2*)

Specific Macro Names

The following macro names have specific meanings:

Macro	Meaning
$*	The target's base name with the extension deleted.
$@	The full name of the current target.
$**	The complete list of dependent files.
$<	The dependent file that is out-of-date with respect to the target (evaluated only for inference rules).
$?	The list of dependents that are out-of-date with respect to the target.
$$@	The target that NMAKE is currently evaluating. A dynamic dependency parameter used only in dependency lines.

$(CC)	The command to invoke the C compiler. By default, NMAKE predefines this macro as CC = cl.
$(AS)	The command that invokes the Microsoft Macro Assembler. NMAKE predefines this macro as AS = masm.
$(MAKE)	The name with which NMAKE is invoked. Used to invoke NMAKE recursively. It causes the line on which it appears to be executed even if the /N option is on. Redefine this macro if you want to execute another program.
$(MAKEDIR)	The directory from which NMAKE was invoked.
$(MAKEFLAGS)	The NMAKE options currently in effect. If you invoke NMAKE recursively, use the command $(MAKE) $(MAKEFLAGS). You cannot redefine this macro.

Inference Rules

Inference rules are templates that NMAKE uses to generate files with a given extension.

Syntax *.fromext.toext: command [[command]] ...*

NMAKE uses these predefined inference rules:

Inference Rule	Command	Default Action
.C.OBJ	$(CC) $(CFLAGS) /C $*.C	CL /C $*.C
.C.EXE	$(CC) $(CFLAGS) $*.C	CL $*.C
.ASM.OBJ	$(AS) $(AFLAGS) $*;	MASM $*;

Directives

The following directives conditionally execute commands, display error messages, include the contents of other files, and turn on or off some of NMAKE's options.

Directive	Description
!IF *expression*	Executes the statements between the **!IF** keyword and the next **!ELSE** or **!ENDIF** directive if *expression* evaluates to a non-zero value. The *expression* consists of integer constants, string constants, or program invocations. Integer constants can use the C unary operators for numerical negation (–), one's complement (~), and logical negation (!), and can also use the C binary operators (+, –, *, /, %, &, |, ^, &&, ||, <<, >>, ==, !=, <, >, <=, and >=).
!ELSE	Executes the statements between the **!ELSE** and **!ENDIF** directives if the statements preceding the **!ELSE** directive were not executed.
!ENDIF	Marks the end of the **!IF**, **!IFDEF**, or **!IFNDEF** block of statements.
!IFDEF *macroname*	Executes the statements between the **!IFDEF** keyword and the next **!ELSE** or **!ENDIF** directive if *macroname* is defined in the description file. NMAKE considers a macro with a null value to be defined.
!IFNDEF *macroname*	Executes the statements between the **!IFNDEF** keyword and the next **!ELSE** or **!ENDIF** directive if *macroname* is not defined in the description file.
!UNDEF *macroname*	Marks *macroname* as being undefined in NMAKE's symbol table.
!ERROR *text*	Causes *text* to be printed, and then stops execution.

!INCLUDE *filename*	Reads and evaluates the file *filename* before continuing with the current description file. If *filename* is enclosed by angle brackets (< >), NMAKE searches for the file in the directories specified by the INCLUDE macro; otherwise it looks in the current directory only. The INCLUDE macro is initially set to the value of the INCLUDE environment variable.
!CMDSWITCHES {+ | −}*opt*	Turns on or off one of four NMAKE options: /D, /I, /N, and /S. If no options are specified, the options are reset to the way they were when NMAKE was started. Turn an option on by preceding it with a plus sign (+), or turn it off by preceding it with a minus sign (−). Using this directive updates the MAKEFLAGS macro.

Pseudotargets

A "pseudotarget" is not a file. It is a name that serves as a handle for building a group of files or executing a group of commands. The NMAKE utility includes these four predefined pseudotargets that provide special rules within a description file.

Pseudotarget	Action
.SILENT :	Does not display lines as they are executed. Has same effect as invoking NMAKE with the /S option.
.IGNORE :	Ignores exit codes returned by programs called from the description file. Has the same effect as invoking NMAKE with the /I option.
.SUFFIXES : *extensions...*	Lists file suffixes for NMAKE to try if it needs to build a target file for which no dependents are specified.
.PRECIOUS : *targets...*	Tells NMAKE not to delete a target if the commands that build it are interrupted. Overrides the NMAKE default.

Summary

The Programmer's WorkBench (PWB) provides an integrated environment for developing programs in C. It runs under OS/2 1.1 and DOS 3.0 and above. With PWB, you can write and edit source and other text files, define development projects, and build applications from one or more files.

Syntax

PWB [[*options*]] [[*files*]]

Options

The options are case sensitive.

/D[[*init*]]	Prevents PWB from examining initialization files, where *init* is one or more of the following characters: T — Ignore TOOLS.INI S — Ignore CURRENT.STS (implies P) P — Ignore current program list If the /D option does not include an *init* character, all three files are ignored.
/e *cmdstr*	Specifies a command on start-up. The entire string of the argument *cmdstr* should be placed in double quotes if it contains a space.
/m *mark*	Moves the cursor to the specified mark instead of moving it to the last known position.
/r	Specifies that PWB starts in read-only mode. Editing of the file is not permitted.
/t [[*file*[[/t[[*file*]]]]...]]	Specifies that any files that follow are temporary. If a single *file* is specified, the editor attempts to load it. If multiple *files* are specified, the first file is loaded. When the *Exit* function is invoked, the editor saves the current file and loads the next file in the list.
/?	Lists the command-line options available for starting PWB.

Arguments to PWB Functions

Arg	Introduces an argument or function. Type ALT+A.
Meta	Modifies the actions of a function. Type F9.
Textarg	A text argument. Type e r g, then the text to be passed to the PWB function.

Numarg	A text argument consisting of characters in the range 0-9.
Markarg	A text argument holding the name of a bookmark defined by the *Mark* function.
Linearg	When PWB is in line mode, entire lines of text are manipulated (deleted, moved, and so on). To select a range of lines, hold down the SHIFT key while moving the cursor.
Streamarg	When PWB is in stream mode, the text is treated as a stream of characters. Any stream of characters can be manipulated.
Boxarg	When PWB is in box mode, columns of text (boxes) can be manipulated.

Categories of PWB Functions

Command Manipulation

Function	Default	Description
Cancel	ESC	Cancels the current operation
Graphic	Most Keys	Inserts the ASCII value of the key into the file
Lastselect	CTRL+U	Recalls the last cursor-movement argument
Lasttext	CTRL+O	Recalls the last *textarg* entered
Quote	CTRL+P	Treats the next keystroke literally
Repeat	—	Repeats the previous command
Undo	ALT+BKSP	Reverses the effect of the last editing change

File Operation

Function	Default	Description
Exit	F8	Exits the editor, with or without saving
Noedit	—	Toggles the no-edit restriction
Refresh	SHIFT+F7	Rereads the file, discarding edits
Saveall	—	Saves all modified files
Setfile	F2	Saves the current file or loads a new file

Cursor Movement

Function	Default	Description
Backtab	SHIFT+TAB	Moves the cursor left to the previous tab stop
Begfile	CTRL+HOME	Moves the cursor to the beginning of file
Begline	HOME	Moves the cursor left to the beginning of the line
Down	DOWN	Moves the cursor down one line
Endfile	CTRL+END	Moves the cursor to the end of the file
Endline	END	Moves the cursor to the right of the last character of the line
Home	NUM 5	Moves the cursor to the upper-left corner of the window
Left	LEFT	Moves the cursor left by one character
Mpage	PGUP	Moves the cursor back by one page
Mpara	—	Moves the cursor back by paragraphs
Mword	CTRL+LEFT	Moves the cursor back by words
Newline	—	Moves the cursor down to the next line
Ppage	PGDN	Moves the cursor forward by one page
Ppara	—	Moves the cursor forward by paragraphs
Pword	CTRL+RIGHT	Moves the cursor forward by words
Right	RIGHT	Moves the cursor right by one character
Tab	TAB	Moves the cursor right to the next tab stop
Up	UP	Moves the cursor up one line

Mark/Goto Position

Function	Default	Description
Mark	CTRL+M	Moves the cursor to the specified position in the file
Restcur	—	Restores the cursor position saved with *Savecur*
Savecur	—	Saves the cursor position for use with *Restcur*

Windows

Function	Default	Description
Maximize	CTRL+F10	Expands the window to its maximum size
Mlines	CTRL+W	Moves the window back by lines
Plines	CTRL+Z	Moves the window forward by lines
Resize	CTRL+F8	Enlarges or shrinks the active window
Setwindow	CTRL+|	Redisplays the window
Window	F6	Creates, removes, or moves between windows

Search/Replace

Function	Default	Description
Mgrep	—	Searches a series of files
Mreplace	—	Replaces throughout a series of files
Msearch	F4	Searches backward
Psearch	F3	Searches forward
Qreplace	CTRL+\	Replaces with confirmation
Replace	CTRL+L	Replaces without confirmation
Searchall	—	Highlights all occurrences of a string

Special Insert

Function	Default	Description
Curdate	-	Inserts the current date (example: 28-Nov-1990)
Curday	—	Inserts the current day (example: Sun)
Curfile	—	Inserts the name of the current file
Curfileext	—	Inserts the extension of the current file
Curfilenam	—	Inserts the base name of the current file
Curtime	—	Inserts the current time (example: 13:45:55)

Insert/Delete

Function	Default	Description
Cdelete	CTRL+G	Deletes the character to the left, excluding line breaks
Copy	CTRL+INS	Copies the lines to the Clipboard
Delete	DEL	Deletes the highlighted area
Emacscdel	BKSP	Deletes the character to the left, including line breaks
Emacsnewl	ENTER	Starts a new line, breaking the current line
Insert	—	Inserts spaces into the highlighted area
Insertmode	INS	Toggles insert mode on and off
Ldelete	CTRL+Y	Deletes lines to the Clipboard
Linsert	CTRL+N	Inserts blank lines
Paste	SHIFT+INS	Inserts text from the Clipboard
Sdelete	—	Deletes a stream of text, including the line breaks
Sinsert	CTRL+J	Inserts blanks, breaking lines if necessary

Programming

Function	Default	Description
Compile	CTRL+F3	Executes the compile or build command
Nextmsg	SHIFT+F3	Moves the cursor to the next error message
Phal	CTRL+]	Balances the parentheses and the brackets

Macro Creation

Function	Default	Description
Assign	ALT+=	Defines macros and sets switches
Execute	F7	Executes a series of editor or macro commands
Message	—	Displays a message on the dialog line
Record	SHIFT+CTRL+R	Turns macro recording on or off
Tell	CTRL+T	Displays an assignment or macro definition
Usercmd	—	Executes a command added to the PWB Run menu

Help

Function	Default	Description
Pwbhelp	—	Displays Microsoft Advisor help screens
Pwbhelpnext	CTRL+F1	Displays the next physical topic in Microsoft Advisor
Sethelp	SHIFT+CTRL+S	Adds or deletes single help files

Miscellaneous

Function	Default	Description
Environment	—	Displays or modifies environment settings
Information	SHIFT+F10	Displays a list of previously edited files
Initialize	SHIFT+F8	Rereads the initialization file
Print	—	Prints all or part of a file
Shell	SHIFT+F9	Spawns a command-line shell

PWB Functions

The PWB functions and the syntax for each are listed alphabetically below. Default keystrokes, if available, are given in parentheses.

Function (Default)	Syntax	Description
Arg (ALT+A)	*Arg*	Introduces a function or an argument for a function.
Assign (ALT+=)	*Assign*	Treats the entire line (except for the line break) on which the cursor is positioned as a function assignment or macro definition.
	Arg Assign	Treats the text from the initial cursor position to the end of the line (not including the line break) as a function assignment or macro definition.
	Arg boxarg Assign	Treats each line of the *boxarg* as an individual function assignment or macro definition. Ignores blank lines and comment lines.
	Arg linearg Assign	Treats each line as a separate function assignment or macro definition, ignoring blank lines and comment lines.
	Arg textarg Assign	Treats *textarg* as a function assignment or macro definition.
	Arg ? Assign	Displays the current function assignments for all functions and macros.
Backtab (SHIFT+TAB)	*Backtab*	Moves the cursor to the previous tab stop. Tab stops are defined to be every *n*th character, where *n* is defined by the **tabstops** switch.
Begfile (CTRL+HOME)	*Begfile*	Places the cursor at the beginning of the file.
Begline (HOME)	*Begline*	Places the cursor on the first nonblank character on the line.
	Meta Begline	Places the cursor in the first character position of the line.
Cancel (ESC)	*Cancel*	Cancels the current operation. If an operation is in progress, it is canceled. If there is no operation, but a message is on the dialog line, the dialog line is restored. If the help window is open, this function closes it.

Function (Default)	Syntax	Description
Cdelete (CTRL+G)	*Cdelete*	Deletes the previous character, excluding line breaks. If the cursor is in column 1, *Cdelete* moves the cursor to the end of the previous line. If issued in insert mode, *Cdelete* deletes the previous character, reducing the length of the line by 1; otherwise, it deletes the previous character and replaces it with a blank. If the cursor is beyond the end of the line when the function is invoked, the cursor is moved to the immediate right of the last character on the line.
Compile (CTRL+F3)	*Compile*	Displays the status of the current compilation (if any) on the dialog line.
	Arg Compile	Compiles and links the current file. Uses the **extmake** command line that matches the file-name extension of the current file.
	Arg textarg Compile	Uses the command line specified by **extmake:text**. The *textarg* replaces %s in the command line.
	Arg Arg textarg Compile	Invokes the specified text as a program. Assumed to display its errors in the format *file row column message*.
	Arg Meta Compile	OS/2 only. Halts a protected-mode compilation running in the background after prompting for confirmation.
Copy (CTRL+INS)	*Copy*	Copies the current line to the Clipboard.
	Arg Copy	Copies text from the initial cursor position to the end of the line and places it in the Clipboard. Ignores the line break.
	Arg boxarg Copy *Arg linearg Copy* *Arg streamarg Copy* *Arg textarg Copy*	Copies the highlighted text to the Clipboard.
	Arg markarg Copy	Copies the range of text between the cursor and the location of the file marker to the Clipboard. In stream mode, a stream of text is selected. In box mode, the text is treated as a *boxarg* or *linearg*, depending on the relative positions of the initial cursor position and the file marker. In line mode, the lines between the cursor and the file marker are copied.

Function (Default)	Syntax	Description
Copy (continued)	*Arg numarg Copy*	Copies the specified number of lines to the Clipboard, starting with the current line.
Curdate	*Curdate*	Inserts the current date at the cursor in the format of 28-Nov-1990.
Curday	*Curday*	Inserts the current day at the cursor in the format of Sun...Sat.
Curfile	*Curfile*	Inserts the fully qualified path name of the current file at the cursor.
Curfileext	*Curfileext*	Inserts the extension of the current file at the cursor.
Curfilenam	*Curfilenam*	Inserts the base name of the current file at the cursor.
Curtime	*Curtime*	Inserts the current time at the cursor in the format of 13:45:55.
Delete (DEL)	*Delete*	Deletes the single character under the cursor, excluding line breaks. SHIFT+DEL places the deleted character in the Clipboard. DEL deletes without copying to the Clipboard.
	Arg Delete	Deletes all text from the current cursor position to the end of the line. The deleted text (including the line break) is placed in the Clipboard. This command has the effect of joining lines.
	Arg boxarg Delete *Arg linearg Delete* *Arg streamarg Delete*	Deletes the highlighted text. The deleted text is placed in the Clipboard.
	Arg Meta Delete *Arg boxarg Meta Delete* *Arg linearg Meta Delete* *Arg streamarg Meta Delete*	Performs the deletions as described above, except the deleted text is not placed in the Clipboard.
	Meta Delete	Deletes the current character or current selection. The deleted text is not copied to the Clipboard.
Down (DOWN or CTRL+X)	*Down*	Moves the cursor down one line. If the cursor moves out of the window, the window is adjusted downward by the number of lines specified by the **vscroll** switch.
	Meta Down	Moves the cursor to the bottom of the window without changing the column position.

Function (Default)	Syntax	Description
Emacsdel (BKSP)	*Emacsdel*	Performs similarly to *Delete*, except that at the beginning of a line while in insert mode, *Emacsdel* deletes the line break between the current line and the previous line, joining the two lines together.
Emacsnewl (ENTER)	*Emacsnewl*	Performs similarly to *Newline*, except that when in insert mode, it breaks the current line at the cursor position.
Endfile (CTRL+END)	*Endfile*	Places the cursor at the end of the file.
Endline (END)	*Endline*	Moves the cursor to the immediate right of the last nonblank character on the line.
	Meta Endline	Moves the cursor one character beyond the column corresponding to the rightmost edge of the window.
Environment	*Environment*	Executes the current line as an environment-variable setting.
	Arg boxarg Environment *Arg linearg Environment*	Executes each highlighted line or line fragment as an environment-variable setting.
	Arg textarg Environment	Executes the text argument as an environment-variable setting.
	Arg ? Environment	Displays all current environment-variable settings.
	Meta Environment	Performs environment "mappings" for all environment variables found on the current line, whenever the variable appears in the following syntax: **$(***environment-variable***)** *or* **$(***environ***:)** For each such environment variable appearing on the line, PWB replaces the variable with the corresponding setting.
	Arg Meta Environment	Performs environment mappings (see description above) for all text from the cursor position to the end of the line.

Function (Default)	Syntax	Description
Environment (*continued*)	*Arg linearg Meta Environment* *Arg boxarg Meta Environment* *Arg streamarg Meta Environment*	Performs environment mappings (see description above) for all highlighted text.
Execute (F7)	*Execute*	Executes a PWB function or macrolist.
	Arg Execute	Treats the line from the initial cursor position to the end as a series of PWB commands, and executes them.
	Arg linearg Execute *Arg textarg Execute*	Treats the specified text as PWB commands and executes them, following the standard rules of macro execution.
Exit (F8)	*Exit*	Exits PWB. If the **autosave** switch is set to yes, the file in memory is automatically saved. If multiple files are specified on the command line, PWB proceeds to load the next file.
	Meta Exit	Performs similarly to *Exit*, except that the current file is not saved.
	Arg Exit	Performs similarly to *Exit*, except that if multiple files are specified on the command line, PWB exits without advancing to the next file.
	Arg Meta Exit	Performs similarly to *Arg Exit*, except that PWB does not save the current file.
Graphic (`)	*Graphic*	Inserts the ASCII value of the specified key into the file.
Home (Keypad 5)	*Home*	Places the cursor in the upper left corner of the current window.
Information (SHIFT+F10)	*Information*	Loads an information file that contains a list of all files in memory along with the current set of files that you have edited. The size of this list is controlled by the **tmpsav** switch, which has a default value of 20.
Initialize (SHIFT+F8)	*Initialize*	Reads all the editor statements from the [PWB] section of TOOLS.INI.
	Arg Initialize	Reads the editor statements from the TOOLS.INI file, using the continuous string of nonblank characters, starting with the initial cursor position as the tag name.

Function (Default)	Syntax	Description
Initialize (*continued*)	*Arg textarg Initialize*	Reads all the editor statements from the [PWB-*textarg*] section of TOOLS.INI.
Insert	*Insert*	Inserts a single blank space at the current cursor position.
	Arg Insert	Inserts a carriage return at the initial cursor position, splitting the line.
	Arg streamarg Insert *Arg linearg Insert* *Arg boxarg Insert*	Inserts blank spaces into the highlighted area.
Insertmode (INS or CTRL+V)	*Insertmode*	Toggles between insert mode and overtype mode.
Lastselect (CTRL+U)	*Lastselect*	Recalls the last cursor-movement argument. This function produces the same result as returning to the last *Arg* position, invoking the *Arg* function, and then recreating the last cursor-movement argument.
Lasttext (CTRL+O)	*Lasttext*	Recalls the last *textarg*. This function produces the same result as typing *arg* once and then retyping the previous *textarg*.
Ldelete (CTRL+Y)	*Ldelete*	Deletes the current line and places it in the Clipboard.
	Arg Ldelete	Deletes text, starting with the initial cursor position through the end of the line, and places it in the Clipboard. Note that it does not join the current line with the next line.
	Arg boxarg Ldelete *Arg linearg Ldelete*	Deletes the specified text from the file and places it in the Clipboard, treating the argument as a *linearg* or *boxarg* regardless of what mode PWB is in.
Left (LEFT or CTRL+S)	*Left*	Moves the cursor one character to the left. If this results in the cursor moving out of the window, the window is adjusted to the left by the number of columns specified by the **hscroll** switch.
	Meta Left	Moves the cursor to the leftmost position in the window on the same line.
Linsert (CTRL+N)	*Linsert*	Inserts one blank line above the current line.
	Arg Linsert	Inserts or deletes blanks at the beginning of a line to make the first nonblank character appear under the cursor.

Function (Default)	Syntax	Description
Linsert (*continued*)	*Arg boxarg Linsert* *Arg linearg Linsert*	Fills the specified area with blanks, treating the argument as a *linearg* or *boxarg* regardless of what mode PWB is in.
Mark (CTRL+M)	*Mark*	Moves the cursor to the beginning of the file.
	Arg Mark	Restores the cursor to its previous location. PWB remembers only the location prior to the last cursor movement.
	Arg numarg Mark	Moves the cursor to the beginning of the specified line, where *numarg* specifies the line number in the file.
	Arg textarg Mark	Moves the cursor to the specified file marker.
	Arg Arg textarg Mark	Defines a file marker at the initial cursor position. If the **markfile** switch contains a file name for saving marks, *textarg* will be added to the file when you exit PWB.
	Arg Arg textarg Meta Mark	Deletes a marker definition.
Maximize (CTRL+F10)	*Maximize*	Expands the window to its maximum size.
	Meta Maximize	Restores the window to its original size.
Message	*Message*	Clears the dialog line.
	Arg textarg Message	Prints the text argument on the dialog line.
	Meta Message	Forces a screen update.
Meta (F9)	*Meta*	Modifies the action of the function it is used with. Refer to the individual functions for specific information.
Mgrep	*Mgrep*	Searches for the previously defined string or pattern. PWB searches all files listed in the **mgreplist** macro.
	Arg Mgrep	Searches files for the string defined as the characters from the initial cursor position to the first blank character.
	Arg textarg Mgrep	Searches files for the specified text.
	Arg Arg Mgrep	Searches files for the regular expression defined as the characters from the initial cursor position to the first blank character.
	Arg Arg textarg Mgrep	Searches files for a regular expression as defined by *textarg*.

Function (Default)	Syntax	Description
Mgrep (*continued*)	*Meta Mgrep* *Arg Meta Mgrep* *Arg textarg Meta Mgrep* *Arg Arg Meta Mgrep* *Arg Arg textarg Meta Mgrep*	Performs similarly to the command form above, except that the value of the **case** switch is temporarily reversed.
Mlines (CTRL+W)	*Mlines*	Moves the window back by the number of lines specified by the **vscroll** switch.
	Arg Mlines	Moves the window until the line that the cursor is on is at the bottom of the window.
	Arg numarg Mlines	Moves the window back by the specified number of lines.
Mpage (PGUP or CTRL+R)	*Mpage*	Moves backward in the file by one window.
Mpara	*Mpara*	Moves the cursor to the first blank line preceding the current paragraph.
	Meta Mpara	Moves the cursor to the last previous line that has text.
Mreplace	*Mreplace*	Performs a search-and-replace across multiple files, prompting for the search and replacement strings, and prompting at each occurrence for confirmation. This function searches all files listed in the **mgreplist** macro.
	Arg Arg Mreplace	Performs the same action as *Mreplace*, but uses regular-expression syntax.
Msearch (F4)	*Msearch*	Searches backward for the previously defined string or pattern. If the string or pattern is found, the window is moved to display it and the matched string or pattern is highlighted. If no match is found, no cursor movement takes place and a message is displayed.
	Arg Msearch	Searches backward in the file for the string defined as the characters from the initial cursor position to the first blank character.
	Arg streamarg Msearch *Arg textarg Msearch*	Searches backward for the specified text.
	Arg Arg Msearch	Searches backward in the file for the regular expression defined as the characters from the initial cursor position to the first blank character.

Function (Default)	Syntax	Description
Msearch (*continued*)	*Arg Arg textarg Msearch*	Searches backward for a regular expression as defined by *textarg*.
	Meta Msearch *Arg Meta Msearch* *Arg textarg Meta Msearch* *Arg Arg Meta Msearch* *Arg Arg textarg Meta Msearch*	Performs similarly to the command form above, except that the value of the **case** switch is temporarily reversed.
Mword (CTRL+LEFT or CTRL+A)	*Mword*	Moves the cursor to the beginning of the current word. If the cursor is not currently in a word or it is at the first character, it moves to the beginning of the previous word.
	Meta Mword	Moves the cursor to the immediate right of the previous word.
Newline (SHIFT+ENTER)	*Newline*	Moves the cursor to a new line. PWB tries to place the cursor in an appropriate position based on the type of file. If the file is a C program and the **softer** switch is set, PWB tries to tab forward, based on continuation of lines and open blocks. If the next line is blank, PWB places the cursor in the column corresponding to the first nonblank character of the previous line. If neither of the above is true, PWB places the cursor on the first nonblank character of the line.
	Meta Newline	Moves the cursor to column 1 of the next line.
Nextmsg (SHIFT+F3)	*Nextmsg*	Advances to the next error message.
	Arg numarg Nextmsg	Moves forward or backward *numarg* error messages. A *numarg* value of 1 moves to the next message; a value of -1 moves back to the previous message.
	Arg Nextmsg	Moves to the next error message (within the current set of messages) that does not refer to the current file.

Function (Default)	Syntax	Description
Nextmsg (*continued*)	*Arg Arg Nextmsg*	Positions the text-file cursor at the line with the error described in the error message at the current cursor position in the <compile> pseudofile. This message becomes the current error message, and the equivalent of *Nextmsg* without any arguments is executed. The following *Nextmsg* command displays the next error message from the <compile> pseudofile.
	Meta Nextmsg	OS/2 only. Advance to the next "set" of error messages, in which a set corresponds to all the error messages for a single compilation. After this command is executed, the previous set is deleted (though you can still view all subsequent sets of error messages in the <compile> pseudofile until it is deleted).
Noedit	*Noedit*	Reverses the no-edit condition, so that if PWB was started with the /R (read-only) option, this command removes the no-edit limitation. If PWB is not in the no-edit state, this command disallows all editing commands that alter a file.
	Meta Noedit	Reverses the no-edit condition for the current file.
Paste (SHIFT+INS)	*Paste*	Inserts the contents of the Clipboard above the current line if the contents were placed there in a line-oriented way, such as with *linearg* or *numarg*. Otherwise, inserts the contents of the Clipboard at the current cursor position.
	Arg streamarg Paste *Arg linearg Paste* *Arg boxarg Paste*	Replaces the specified text with the contents of the Clipboard.
	Arg Paste	Inserts the text from the initial cursor position to the end of the line at the initial cursor position.
	Arg textarg Paste	Places the specified text in the Clipboard and inserts that text at the initial cursor position.

Function (Default)	Syntax	Description
Paste (*continued*)	*Arg Arg textarg Paste*	Interprets *textarg* as a file name and inserts the contents of that file into the current file above the current line.
	Arg Arg !textarg Paste	Treats the text as a DOS or OS/2 command and inserts its output to **stdout** into the current file at the initial cursor position.
Phal (CTRL+[)	*Phal*	Scans backward through the file, balancing parentheses and brackets. The first unbalanced one is highlighted when found. If it is found and is not visible, PWB displays the matching line on the dialog line, with the highlighted matching character. The corresponding character is placed into the file at the current cursor position. Note that the search does not include the current cursor position and that the scan looks only for more left brackets or parentheses than right, not just for an unequal number.
	Arg Phal	Performs similarly to *Phal*, except that it scans forward in the file and looks for more right brackets or parentheses than left.
	Meta Phal	Performs similarly to *Phal*, except that the file is not updated.
	Arg Meta Phal	Performs similarly to *Arg Phal*, except that the file is not updated.
Plines (CTRL+Z)	*Plines*	Adjusts the cursor forward by the number of lines specified by the **vscroll** switch.
	Arg Plines	Moves the cursor downward so the line that the cursor is on is at the top of the window.
	Arg numarg Plines	Moves the cursor forward the specified number of lines.
Ppage (PGDN or CTRL+C)	*Ppage*	Moves forward in the file by one window.
Ppara	*Ppara*	Moves the cursor forward one paragraph and places the cursor on the first line of the new paragraph.
	Meta Ppara	Moves the cursor to the first blank line following the current paragraph.

Function (Default)	Syntax	Description
Print	*Print*	Prints the current file. If the **printcmd** switch is set, this function uses the system-level command given in the switch. Otherwise, the function copies output to LPT1.
	Arg textarg Print	Prints all the files listed in the text argument. File names should be separated with a space. Macro file lists are allowed.
	Arg linearg Print *Arg boxarg Print* *Arg streamarg Print*	Prints the highlighted area.
Prompt	*Prompt*	Allows macros to prompt for text arguments.
	Arg Prompt	Uses the text from the cursor to the end of the line as the prompt.
	Arg textarg Prompt	Uses the specified text argument as the prompt.
	Arg linearg Prompt *Arg boxarg Prompt* *Arg streamarg Prompt*	Uses the selected text or the first line of the selected area as the prompt.
	Meta Prompt	Uses the selected text or the first line of the selected area as the prompt and passes it to the next function.
Psearch (F3)	*Psearch*	Searches forward for the previously defined string or pattern. If the string or pattern is found, the window is moved to display it and the matched string or pattern is highlighted. If it is not found, the cursor does not move and a message is displayed.
	Arg Psearch	Searches forward in the file for the string defined as the characters from the initial cursor position to the first blank character.
	Arg textarg Psearch	Searches forward for the specified text.
	Arg Arg Psearch	Searches forward in the file for the regular expression defined as the characters from the initial cursor position to the first blank character.

Function (Default)	Syntax	Description
Psearch (*continued*)	*Arg Arg textarg Psearch*	Searches forward for a regular expression as defined by *textarg*.
	Meta Psearch *Arg Meta Psearch* *Arg textarg Meta Psearch* *Arg Arg Meta Psearch* *Arg Arg textarg Meta Psearch*	Performs similarly to the command form above, except that the value of the **case** switch is temporarily reversed.
Pwbhelp	*Pwbhelp*	Microsoft Advisor help screens.
	Arg Pwbhelp	Displays help on the topic at the cursor position.
	Arg single-line-cursor-movement Pwbhelp	Displays help on the text argument highlighted on the screen.
	Arg textarg Pwbhelp	Displays help on the specified text argument.
	Meta Pwbhelp	Prompts the user for a keystroke and displays help on the function associated with that keystroke. For this command to work, the cursor cannot be resting on a help topic.
Pwbhelpnext (CTRL+F1)	*Pwbhelpnext*	Displays the next physical topic in the Microsoft Advisor help system.
	Meta Pwbhelpnext	Displays the previous help topic on the backtrace list.
	Arg Pwbhelpnext	Displays the next occurrence of the current help topic within the Microsoft Advisor help system.
Pword (CTRL+RIGHT or CTRL+F)	*Pword*	Moves the cursor forward to the beginning of the next word.
	Meta Pword	Moves the cursor to the immediate right of the current word or, if the cursor is not in a word, to the right of the next word.
Qreplace (CTRL+\)	*Qreplace*	Performs a search-and-replace, prompting for the search and replacement strings, and prompting at each occurrence for confirmation. The search begins at the cursor position and continues through the end of the file.
	Arg boxarg Qreplace *Arg linearg Qreplace* *Arg streamarg Qreplace*	Performs the search-and-replace within the highlighted area, prompting at each occurrence for confirmation.

Function (Default)	Syntax	Description
Qreplace (*continued*)	*Arg markarg Qreplace*	Performs the search-and-replace between the cursor position and the specified file marker, prompting at each occurrence for confirmation.
	Arg numarg Qreplace	Performs the search-and-replace over the specified number of lines, starting with the current line, prompting at each occurrence for confirmation.
	Arg Arg Qreplace *Arg Arg boxarg Qreplace* *Arg Arg linearg Qreplace* *Arg Arg streamarg Qreplace* *Arg Arg markarg Qreplace* *Arg Arg numarg Qreplace*	Performs the same as the corresponding command listed above, except that the search pattern is a regular expression and the replacement pattern can select special tagged sections of the search for selective replacement.
Quote (CTRL+P)	*Quote*	Reads one keystroke from the keyboard and treats it literally.
Record (SHIFT+CTRL+R)	*Record*	Toggles the macro recording. When a recording is stopped, PWB assigns all the recorded commands to the default macro name RECORDVALUE. During the recording, the name of each command is written to the <record> pseudofile, which can be placed in a window and viewed as it is dynamically updated.
	Arg textarg Record	Toggles the macro recording and names the macro.
	Meta Record	Toggles the macro recording, but no editing commands are executed until the recording is turned off.
	Arg Record *Arg Arg textarg Record* *Arg Arg Meta Record*	Performs identically to the corresponding command listed above, but appends editing commands to the end of the macro.
Refresh (SHIFT+F7)	*Refresh*	Asks for confirmation and then rereads the file from disk, discarding all edits since the file was last saved.
	Arg Refresh	Asks for confirmation and then discards the file from memory and from the file history. The most recently edited previous file becomes the current file.

Function (Default)	Syntax	Description
Repeat	*Repeat*	Repeats the last editing command, using precisely the same arguments and *Meta* condition used by the last command. However, the command is executed relative to the new cursor position.
	Arg numarg Repeat	Performs the last editing command for the number of times specified by *numarg*. Note that *Repeat* must be assigned to a key to execute this command.
Replace (CTRL+L)	*Replace*	Performs a search-and-replace without confirmation, prompting for the search string and replacement string. The search begins at the cursor position and continues through the end of the file.
	Arg boxarg Replace *Arg linearg Replace* *Arg streamarg Replace*	Performs the search-and-replace in the highlighted area.
	Arg markarg Replace	Performs the search-and-replace between the cursor and the specified file marker.
	Arg numarg Replace	Performs the search-and-replace over the specified number of lines, starting with the current line.
	Arg Arg Replace *Arg Arg boxarg Replace* *Arg Arg linearg Replace* *Arg Arg streamarg Replace* *Arg Arg markarg Replace* *Arg Arg numarg Replace*	Performs the same as the corresponding command listed above, except that the search pattern is a regular expression and the replacement pattern can select special tagged sections of the search for selective replacement.
Resize (CTRL+F8)	*Resize*	Enlarges or shrinks the active window. You must have at least two windows open to use this function.
Restcur	*Restcur*	Restores the cursor position saved with *Savecur*.
Right (RIGHT or CTRL+D)	*Right*	Moves the cursor one character to the right. If this would move the cursor off the screen, the window moves right by the number of columns specified by the **hscroll** switch.
	Meta Right	Moves the cursor to the window's rightmost column.
Saveall	*Saveall*	Saves to disk all files that have been altered but not yet saved.

Function (Default)	Syntax	Description
Savecur	*Savecur*	Saves the current cursor position. Restores it with *Restcur*.
Sdelete	*Sdelete*	Deletes the single character under the cursor, excluding line breaks. It does not place the deleted character in the Clipboard.
	Arg Sdelete	Deletes from the current line at the point of the cursor position. The text deleted (including the line break) is placed in the Clipboard.
	Arg streamarg Sdelete	Deletes the stream of text from the initial cursor position up to the current cursor position and places it in the Clipboard, regardless of the current selection mode.
Searchall	*Searchall*	Highlights all occurrences of the previously defined string or pattern. If the string exists in the file, the cursor moves to the first occurrence.
	Arg Searchall	Highlights all occurrences of the string defined as the characters from the initial cursor position to the first blank character.
	Arg textarg Searchall	Highlights all occurrences of the specified text.
	Arg Arg Searchall	Highlights all occurrences of the regular expression defined as the characters from the initial cursor position to the first blank character.
	Arg Arg textarg Searchall	Highlights all occurrences of a regular expression as defined by *textarg*.
	Meta Searchall *Arg Meta Searchall* *Arg textarg Meta Searchall* *Arg Arg Meta Searchall* *Arg Arg textarg Meta Searchall*	Performs similarly to command above, except that the value of the **case** switch is temporarily reversed.
Selcur	*Selcur*	Selects a portion of text (box, stream, or line) from the current cursor position to an anchor marked by the *Savecur* function.
Selmode	*Selmode*	Toggles between stream mode, line mode, and box mode.
Select (SHIFT+DOWN)	*Select*	In combination with a cursor-movement key, selects a box, stream, or line of text to be passed to another PWB function.

Function (Default)	Syntax	Description
Setfile (F2)	*Setfile*	Switches to the previously edited file, saving to disk any changes made to the current file if the **autosave** switch is set to yes.
	Arg Setfile	Switches to the file name under the cursor beginning at the initial cursor position.
	Arg textarg Setfile	Switches to the file specified by *textarg*. If the text argument is a drive or directory, PWB changes the current drive or directory.
	Meta Setfile *Arg Meta Setfile* *Arg textarg Meta Setfile*	Performs similarly to the corresponding command listed above, but does not save the changes made to the current file.
	Arg Arg textarg Setfile	Saves the current file under the name specified by *textarg*.
	Arg Arg Setfile	Saves the current file.
Sethelp (SHIFT+CTRL+S)	*Sethelp*	Adds or deletes individual help files from the list of files PWB searches for on-line help. Unlike the **helpfiles** switch, which lists the complete set of help files to be used, *Sethelp* adds or removes individual files from the help-file list without affecting the rest of the list. *Sethelp* affects only the current PWB session.
	Arg Sethelp	Adds the file name under the current cursor position to the list of help files searched.
	Arg streamarg Sethelp	Adds the highlighted file name to the list of files searched.
	Arg textarg Sethelp	Adds the file name specified by *textarg* to the list of help files searched.
	Arg ? Sethelp	Lists all currently open help files.
	Arg Meta Sethelp	Removes the file name under the current cursor position from the list of help files searched.
	Arg Streamarg Meta Sethelp	Removes the highlighted file name from the list of files searched.
	Arg textarg Meta Sethelp	Removes the file name specified as *textarg* from the list of files searched.

Function (Default)	Syntax	Description
Setwindow (CTRL+])	*Setwindow*	Redisplays the entire screen.
	Arg Setwindow	Adjusts the window so that the initial cursor position becomes the home position (upper left corner).
	Meta Setwindow	Redisplays the current line.
Shell (SHIFT+F9)	*Shell*	Saves the current file if the **autosave** switch is set to yes and runs the command shell.
	Meta Shell	Runs the command shell without saving the current file.
	Arg Shell	Uses the text on the screen from the cursor up to the end of the line as a command to the shell.
	Arg boxarg Shell *Arg linearg Shell*	Treats each line of either argument as a separate command to the shell.
	Arg textarg Shell	Uses *textarg* as a command to the shell.
Sinsert (CTRL+J)	*Sinsert*	Inserts a single blank space at the current cursor position.
	Arg Sinsert	Inserts a carriage return at the initial cursor position, splitting the line.
	Arg streamarg Sinsert	Inserts a stream of blanks between the initial cursor position and the current cursor position.
Tab (TAB)	*Tab*	Moves the cursor to the next tab stop. Tab stops are defined by the **tabstops** switch.
Tell (CTRL+T)	*Tell*	Prompts for a keystroke, then displays the name of the keystroke and the function assigned to it in the format *function:keyname*.
	Arg Tell	Identical to *Tell*, but if the keystroke has a macro attached, displays *MacroName:=MacroValue*.
	Arg Arg Tell	Prompts for a keystroke, then displays the value of the macro attached to the key. If a function is assigned to the key, PWB displays the name of the function.
	Arg textarg Tell	Performs like *Arg Tell* but obtains the macro name from a *textarg* rather than a keystroke.

Function (Default)	Syntax	Description
Tell (continued)	*Meta Tell* *Arg Meta Tell* *Arg Arg Meta Tell* *Arg textarg Meta Tell*	Performs the same as the command listed above, except PWB inserts the output into the file rather than on the dialog line.
Undo (ALT+BKSP)	*Undo*	Reverses the last editing change. The **undocount** switch (default value: 10) sets the limit for the *Undo* command.
	Meta Undo	Recalls a command previously canceled with *Undo*. This command is often called "redo."
Up (UP or CTRL+E)	*Up*	Moves the cursor up one line. If this would move the cursor out of the window, the window moves upward by the number of lines specified by the **vscroll** switch.
	Meta Up	Moves the cursor to the top of the window, retaining the column position.
Usercmd	*Usercmd*	Executes a command added to the PWB Run menu by the user.
	Arg numarg Usercmd	Executes the given user-customized command from the Run menu. The *numarg* can be '1', '2', '3', '4', '5', or '6'.
Window (F6)	*Window*	Moves the cursor to the next window.
	Arg Window	Splits the current window horizontally at the cursor position. All windows must contain at least five lines.
	Arg Arg Window	Splits the current window vertically at the cursor position. All windows must contain at least ten columns.
	Meta Window	Closes the window.
Winstyle (CTRL+F6)	*Winstyle*	Toggles between window styles: two scroll bars, horizontal only, vertical only, or no scroll bars.

Return Values of PWB Functions

The return values of PWB functions are useful in conditional macros.

Function	Returns TRUE	Returns FALSE
Arg	Always	Never
Assign	Assignment successful	Invalid assignment
Backtab	Cursor moved	Cursor at left margin
Begfile	Cursor moved	Cursor not moved
Begline	Cursor moved	Cursor not moved
Cancel	Always	Never
Cdelete	Cursor moved	Cursor not moved
Compile	Compilation successfully initiated, or background compilation running	Compilation unsuccessfully initiated, or background compilation not running
Copy	Always	Never
Curdate	Date inserted	Insertion would make line too long
Curday	Day inserted	Insertion would make line too long
Curfile	File inserted	Insertion would make line too long
Curfileext	File extension inserted	Insertion would make line too long
Curfilenam	File name inserted	Insertion would make line too long
Curtime	Time inserted	Insertion would make line too long
Delete	Always	Never
Down	Cursor moved	Cursor not moved
Emacsdel	Cursor moved	Cursor not moved
Emacsnewl	Always	Never
Endfile	Cursor moved	Cursor not moved
Endline	Cursor moved	Cursor not moved
Environment	Successful set or map	Syntax error or line too long
Execute	Last command successful	Last command failed
Exit	No return condition	No return condition
Graphic	Character inserted	Insertion would make line too long
Home	Cursor moved	Cursor not moved
Information	Always	Never
Initialize	Found tagged section in TOOLS.INI	Did not find tagged section in TOOLS.INI
Insert	Always	Never

Function	Returns TRUE	Returns FALSE
Insertmode	Insert mode turned on	Insert mode turned off
Lastselect	Selection re-created	*Arg* already selected
Lasttext	Value of function eventually executed	Bad argument
Ldelete	Always	Never
Left	Cursor moved	Cursor not moved
Linsert	Always	Never
Mark	Definition or move successful	Bad argument, or specified file marker not found
Message	Always	Never
Meta	*Meta* turned on	*Meta* turned off
Mgrep	String found	String not found or specified, or search terminated by CTRL+BREAK, or background compilation in progress
Mlines	Movement occurred	Bad argument
Mpage	Movement occurred	Bad argument
Mpara	Cursor moved	Cursor not moved
Mreplace	Replacement successful	Replacement failed, or was aborted
Msearch	String found	Bad argument, or string not found
Mword	Cursor moved	Cursor not moved
Newline	Always	Never
Nextmsg	Message found	No more messages found
Noedit	File or editor in no-edit state	File or editor is editable
Paste	Usually	Tried *Arg Arg filename Paste* and file did not exist
Pbal	Balance successful	Bad argument, or string not found
Plines	Movement occurred	Bad argument, or not balanced
Ppage	Cursor moved	Cursor not moved
Ppara	Cursor moved	Cursor not moved
Print	Print successfully submitted	Could not start print job
Psearch	String found	Bad argument, or string not found
Pword	Cursor moved	Cursor not moved
Qreplace	At least one replacement	String not found, or invalid pattern
Quote	Almost always	Insertion would make line too long
Record	Recording turned on	Recording turned off

Function	Returns TRUE	Returns FALSE
Refresh	File read in or deleted	Canceled, or bad argument
Repeat	Function repeated and returned TRUE	Function repeated and returned FALSE, or no function to repeat
Replace	At least one replacement	String not found, or invalid pattern
Restcur	Position previously saved with *Savecur*	Position not saved with *Savecur*
Right	Cursor on the text of a line	Cursor beyond end of line
Saveall	Always	Never
Savecur	Always	Never
Sdelete	Always	Never
Searchall	Something found	Nothing found
Setfile	File switch successful	No alternate file, or current file needs to be saved and can't be saved
Setwindow	Always	Never
Shell	Shell successful	Bad argument, or program not found
Sinsert	Always	Never
Tab	Cursor moved	Cursor not moved, or program not found
Tell	Key pressed has function assigned	Key pressed has no function assigned
Undo	Usually	If nothing to undo
Up	Cursor moved	Cursor not moved
Window	Successful split, join, or move	Any error

PWB Switches

To set a numeric or text switch, invoke the Options menu Editor Settings command. Within the list of PWB settings, switches are followed by a colon and the current setting:

switch:value

where *switch* is the name of the switch and *value* is a string of digits (numeric switch) or a string of text (text switch).

After modifying the switch, save the PWB settings file with the File menu Save command.

The following table lists switches, along with their type, default value, and description.

Switch	Type	Default Value	Description
askexit	Boolean	No	Prompts for confirmation when you exit from PWB.
askrtn	Boolean	Yes	Prompts you to press ENTER when returning from a *Shell* command.
autosave	Boolean	Yes	Saves the current file whenever you move to another. If this switch is set to no, contents of file buffer are maintained, but subsequent actions, such as exiting, may lose edits.
backup	Text	bak	Determines what happens to the old copy of a file when a new version is saved to disk. A value of none specifies that no backup operation is to be performed: PWB overwrites the old file. A value of undel specifies that the old file is to be moved to the DELETED directory so that UNDEL.EXE can retrieve it. A value of bak specifies that the file name of the old version of the file will be changed to .BAK.
browwindow	Boolean	Yes	Defines how PWB displays Source Browser information. If the switch is set to yes, Source Browser information appears in a split window. If the switch is set to no, Source Browser information overwrites the current window.
build	Text	—	Defines a command line that PWB uses when you invoke the *Compile* function or select Compile and Build options from the Options menu. **Build** switches support many of the same features as the NMAKE utility. PWB uses the **build** switch to define the following features for a program list (in order of appearance in the <assign> pseudofile):

Switch	Type	Default Value	Description
build *(continued)*			- NMAKE macros used in the program list - Build targets set during the current session - Build-target commands: the combination of commands, defined in other build switches, used to build the targets - NMAKE inference rules used in the program list - Commands used in build-target commands for release and debug builds - Files to be included in the program list - Error messages to be displayed if the build fails Generally, you do not need to set the **build** switch directly, since you can customize most of your compile and link operations by selecting commands from the Options menu. However, you may want to set the **build** switch directly to: - Compile languages that are not available on the Options menu - Use options not listed on the Options menu - Perform complicated builds not supported by the Options menu
case	Boolean	No	Considers case to be significant for search and replace operations. For example, if **case** is set to yes, the string `Procedure` is not found as a match for the string `procedure`.

Switch	Type	Default Value	Description
color	Text	—	Specifies colors for windows and menus. A table of each color name, default value, and description follows this table.
dblclick	Numeric	—	Sets the mouse double-click threshold, the maximum time between mouse button clicks for them to be considered a double-click. In DOS, the units are measured in units of 1/18 second, under OS/2 the units are milliseconds.
deflang	Text	—	Defines the language and the default extension used for file names. Possible values include BASIC (*.BAS), C (*.C), COBOL (*.CBL), FORTRAN (*.FOR), Lisp (*.LSP), MASM or ASM (*.ASM), Pascal (.PAS), and NONE (*.*).
defwinstyle	Numeric	7	Defines the window style and the presence or absence of the border and scroll bars. Possible values include: 0 No border, no scroll bars 1 Border, no scroll bars 3 Border, vertical scroll bar only 5 Border, horizontal scroll bar only 7 Border, vertical and horizontal scroll bars
doslibs	Text	Override	Tells PWB whether DOS/real-mode libraries have been installed and, if so, whether they have default or explicit names. Possible values include "none" (not installed), "default" (installed with default names), or "override" (installed with explicit names).
edit	Boolean	Yes, unless the /r command-line option has been specified	Allows files to be edited. When **edit** is set to no, the file cannot be changed.

Switch	Type	Default Value	Description
editreadonly	Boolean	Yes	Allows disk files marked as read-only to be edited. If the switch is set to no, a read-only file is marked no-edit in PWB.
entab	Numeric	1	Controls the degree to which PWB converts multiple spaces to tabs when editing a file. A value of 0 means tabs are not used to represent white space; 1 means all multiple spaces outside of quoted strings are converted; 2 means all multiple spaces are converted to tabs.
enterinsmode	Boolean	Last state of insert or over-type mode from previous editing session	Starts up PWB in insert mode instead of overtype mode, or switches to insert mode in the middle of an editing session.
enterselmode	Text	Stream mode	Determines the start-up selection mode for PWB: line, box, or stream.
factor	Numeric	—	With the **friction** switch factor: %10 5 means "reduce the friction by a factor of 10% every 5 repeated keystrokes," and factor: –60 12 means "reduce the friction by a constant 60 every 12 repeated keystrokes."
fastfunc	Text	—	Enables auto-repeating for PWB functions. By default, the functions that auto-repeat are *Up*, *Down*, *Left*, *Right*, *Mlines*, *Mpage*, *Mpara*, *Mword*, *Plines*, *Ppage*, *Ppara*, and *Pword*. The speed at which a function auto-repeats is controlled by the **friction** and **factor** switches.

Switch	Type	Default Value	Description
filetab	Numeric	8	Determines how PWB translates tabs to spaces when reading or writing a disk file. This switch also determines how PWB translates spaces to tabs for modified lines, when **entab**> 0. The value of the switch gives the number of spaces associated with each tab column. For example, the setting "filetab:4" assumes a tab column every four positions on each line. Every time PWB finds a tab character in a file, it loads the buffer with the number of spaces necessary to get to the next tab column.
friction	Numeric	—	With the **factor** switch, controls the amount of repeat rate (friction) of the direction keys. A larger value means the repeat rate is slower.
height	Numeric	25	Controls the number of lines in the PWB editing window. This is useful with a nonstandard display device. Enhanced Graphics Adapter (EGA) in 43-line mode on the IBM PC uses a value of 43. VideoGraphics Array (VGA) in 50-line mode uses a value of 50.
helpfiles	Text	$PATH: PWB.HLP	Specifies which .HLP files should be searched by on-line help.
helplist	Boolean	Yes	If the switch is set to yes and the user requests help on a topic for which there are several entries, PWB displays a list of available help files. If the switch is set to no or if there is only one help entry for the topic, PWB displays the first information it finds about a given topic.
helpwindow	Boolean	Yes	Controls split-screen behavior. When the switch is set to no, PWB does not split the screen to display on-line help information.

Switch	Type	Default Value	Description
hike	Numeric	4	Specifies the cursor's new-line position (from the top of the screen) when the cursor is moved out of the current window by more than **vscroll** lines.
hscroll	Numeric	10	Controls the number of columns shifted left or right when the cursor is scrolled out of the editing window.
load	Text	—	Specifies the name of a PWB extension executable file to be loaded. When this switch is assigned a new value, the extension file named is loaded into memory and initialized by calling its **WhenLoaded** function.
markfile	Text	—	Specifies the name of the file PWB searches when looking for a marker that is not in the in-memory set. This file can be created by entering lines of the following form: *markername filename line column* Here, *line* and *column* specify the position in the file *filename* where the marker *markername* appears.
menukey	Text	ALT	Defines the key that enables the PWB menu bar.
msgflush	Boolean	Yes	Determines the fate of previous compiler messages when a background compile operation occurs. When the switch is set to yes, the old compiler messages are deleted. When it is set to no, the old messages can be retained. Under DOS, compiler messages are always cleared, because background compiles are impossible.
noise	Numeric	50	Controls the number of lines counted at a time when searching or loading a file. This value is displayed in the lower right corner of the screen or in a status box and can be set to no by setting **noise** to 0.

Switch	Type	Default Value	Description
os2libs	Text	Override	Tells PWB whether protected-mode libraries have been installed and, if so, whether they have default or explicit names. Possible values include "none" (not installed), "default" (installed with default names), or "override" (installed with explicit names).
printcmd	Text	—	Specifies a system-level command that PWB invokes when you issue the *Print* command.
readonly	Text	-	Specifies the DOS command invoked when PWB attempts to overwrite a read-only file. The current file name replaces %s.
realtabs	Boolean	Yes	Preserves actual tab characters instead of converting them to spaces. When this switch is set to yes, PWB preserves tab alignment as characters are added and deleted.
rmargin	Numeric	72	Controls the right column margin used for wordwrap mode. Any character typed to the right of this margin causes a line break. Wordwrap mode is turned on and off with the **wordwrap** switch.
savescreen	Boolean	Yes	Saves and restores the DOS or OS/2 screen (used with the *Shell* and *Exit* functions).
searchwrap	Boolean	No	Causes search and replace commands to wrap past the end of the file and continue searching from the beginning. Unsuccessful searches stop after the entire file is searched once. When this switch is set to no, searches stop at the beginning or end of the file.
shortnames	Boolean	Yes	Allows you to load a file by giving only the base name, which PWB searches for in the <information file>.

Switch	Type	Default Value	Description
softer	Boolean	Yes	Attempts to indent, based upon the format of the surrounding text when you invoke the *Newline* or *Emacsnewl* function.
tabalign	Boolean	No	Determines where the cursor can be placed in a tab field. When the switch is set to no, the cursor can be placed anywhere in a tab field. When the switch is set to yes (and if **realtabs** is also set to yes), the cursor aligns with the column position of the tab.
tabdisp	Numeric	32	Specifies the ASCII value of the TAB character. Normally, a space is used, but a graphic character can be used to show which spaces correspond to tabs.
tabstops	Numeric	4	Controls the number of spaces between each logical tab stop for the *Tab* and *Backtab* movement functions. This switch has no relation to the interpretation of actual tabs.
tmpsav	Numeric	20	Controls the maximum number of files about which information is kept between editing sessions. Each file is listed once. When you exit PWB, the positions of the cursor and window are saved. When you edit a file a second time, the screen is restored to its previous condition. If this switch is set to 0, PWB lets the information file grow without limit; all files ever edited appear in the information file until CURRENT.STS is altered or deleted.
traildisp	Numeric	0	Specifies the ASCII value of the character to be displayed as trailing spaces. This switch has no effect unless the **trailspace** switch is set to yes.

Switch	Type	Default Value	Description
traillines	Boolean	No	If this switch is set to yes, blank lines at the end of the file are retained when the file is saved. If the switch is set to no, PWB does not save any blank lines following the last nonblank line.
traillinesdisp	Numeric	Space ASCII 32	Specifies the character displayed in the first column of each line beyond the end of a file.
trailspace	Boolean	No	Preserves trailing spaces in each line you modify.
undelcount	Numeric	No limit	Controls the number of backup copies of a file saved when the **backup** switch is set to **undel**. When the limit is exceeded, PWB discards the oldest backup.
undocount	Numeric	10	Controls the number of times you can undo or redo an edit function.
unixre	Boolean	Yes	Specifies the UNIX regular-expression syntax rather than the syntax used by PWB.
user	Text	—	Customizes the PWB Run menu by adding a program that can be called from within PWB. The **user** switch takes these parameters: **user**: title, path, arg, output, dir, helpline, runtime, return, backgrd, key The first two parameters are mandatory. *Title* is the name of the program as it appears on the menu, and *path* is the complete file specification.

Switch	Type	Default Value	Description
user (*continued*)			The remaining parameters are optional:
		—	*arg*: command-line arguments
		—	*output*: output file
		Current directory	*dir*: working directory
		None	*helpline*: Appears on left side of status bar
		—	*runtime*: Determines whether PWB displays a run-time dialog box for passing arguments to the program
		No	*return*: Determines whether PWB is to prompt for a keystroke upon completion
		No	*backgrd*: Determines whether an OS/2 program runs in the background
			key: Assigns a shortcut key to this program (1-9)
viewonly	Boolean	No	When set to yes, the file in memory can be viewed, but not edited.
vscroll	Numeric	7	Controls the number of lines shifted up or down when the cursor is scrolled out of the editing window. The *Mlines* and *Plines* functions also use this value.
width	Numeric	80	Controls the width of the display mode for displays that are capable of showing more than 80 columns. Values other than 80 are supported only for a limited number of monitors.
winlibs	Text	Override	Tells PWB whether Windows libraries have been installed and, if so, whether they have default or explicit names. Possible values include "none" (not installed), "default" (installed with default names), or "override" (installed with explicit names).
wordwrap	Boolean	No	Breaks lines of text when you edit them beyond the margin specified by **rmargin**.

Color Switch Values

The following table lists the color names used with the color switch, along with the default value and description of each. The syntax

color:*option*

is used.

Color Names	Description	Default Value
alert	MessageBox alerts	70
border	Window borders	07
buttondown	Pushed button color	07
dialogaccel	Dialog box accelerators	7f
dialogaccelbor	Dialog box accelerator border	7f
dialogbox	Dialog box background	70
disabled	Disabled items	78
elevator	Scroll bar elevator	07
enabled	Enabled items	70
helpbold	Help boldface text	1f
helpitalic	Help italic text	1a
helpnorm	Help normal text	17
helpunderline	Help underline text	1c
helpwarning	Help warning text	70
hilitectrl	Highlighted control items	07
info	Status line without buttons	3f
itemhilitesel	One highlighted character (under selection bar)	0f
listbox	Listbox background	70
menu	Menu bar color	70
menubox	Background for menus	70
menuhilite	One highlighted character	7f
menuhilitesel	One highlighted character (under selection bar)	0f
menuselected	Selected menus	07
pushbutton	Pushbutton color	70

Color Names	Description	Default Value
scrollbar	Scroll bar background and arrows	70
selection	Area highlighted by "Arg cursor-movement"	71
shadow	Shadows	08
status	Status information	70
text	Normal text color	17

Regular Expressions

A "regular expression" is a special search string that matches a pattern of text rather than a specific sequence of characters. PWB supports two versions of regular-expression syntax: UNIX® and Microsoft Editor version 1.0. The **unixre** switch controls which version is in effect.

UNIX Special Characters

The following list describes the UNIX special characters. All characters should be interpreted literally, except *class*, which has a special meaning (defined below).

Simple

\	Escape. Causes PWB to ignore the special meaning of the next character.
.	Wild card. Matches any single character.
^	Beginning of line. Matches the following pattern only when it occurs at the beginning of a line.
$	End of line. Matches the preceding pattern only when it occurs at the end of a line.
[*class*]	Character class. Matches any one character in the class. A dash (–) specifies a contiguous range of ASCII values. For example, `[a-zA-Z0-9]` matches any letter or digit, and `[abc]` matches `a`, `b`, or `c`.

	[^*class*]	Inverse of character class. Matches any character not specified in the class.
Complex	*X**	Repeat operator. Matches zero or more occurrences of *X*, where *X* represents a regular expression that is either a single character, a group of characters enclosed in brackets ([ ]), or the regular-expression delimiters '\(' and '\)'. This operator always matches as many characters as possible.
	\(... \)	Tagged expression. A marked substring that you can refer to elsewhere in the search string or in a replacement string. When a tagged expression is referred to in a search string, PWB finds text with the tagged expression repeated. When a tagged expression is referred to in a replacement string, PWB reuses part of the text it is replacing. Characters falling between '\(' and '\)' are treated as a group.
	n	Reference to the characters matched by a tagged expression. The one-digit number *n* indicates which expression.

Non-UNIX Special Characters

The following list describes the special characters originally used in the Microsoft Editor version 1.0. The term *class* has a special meaning (defined below). All other characters should be interpreted literally.

Simple	\	Escape. Causes PWB to ignore the special meaning of the next character.
	?	Wild card. Matches any single character.
	^	Beginning of line. Matches the following pattern only when it occurs at the beginning of a line.
	$	End of line. Matches the preceding pattern only when it occurs at the end of a line.
	[*class*]	Character class. Matches any one character in the class. A dash (–) specifies a range of values. For example, `[a-zA-Z0-9]` matches any character or digit, and `[abc]` matches `a`, `b`, or `c`.

	[~*class*]	Inverse of character class. Matches any character not specified in the class.
Complex	*X**	Minimal matching. Matches zero or more occurrences of *X*, where *X* represents a regular expression that is either a single character or a group of characters enclosed in parentheses (( )), brackets ([ ]), or braces ({ }).
	X+	Minimal matching plus (shorthand for *XX**). Matches one or more occurrences of *X*.
	X@	Maximal matching. Identical to *X**, except that it matches as many characters as it can.
	X#	Maximal matching plus. Identical to *X+*, except that it matches as many characters as it can.
	(*X1*!*X2*!...!*Xn*)	Alternation. Matches either *X1*, *X2*, or *Xn*. It tries to match the expressions in order, and switches from *Xi* to *Xi+1* only if the rest of the expression fails to match.
	~*X*	The not function. Matches nothing, but checks to see whether the string matches *X* at this point and fails if it does.
	X^*n*	Power function. Matches exactly *n* copies of *X*.
	{...}	Tagged expression, which is a string of characters you identify so that you can refer to them elsewhere as $*n*. By referring to a tagged expression in a search string, you cause PWB to look for patterns involving duplication. By referring to a tagged expression in a replacement string, you cause PWB to reuse part of the text that it is replacing. Characters within braces are treated as a group.
	$*n*	Reference to a previously tagged substring. The number *n* indicates which substring. The first tagged substring is represented as $1, the second as $2, and so on. The tag $0 represents the entire matched string.
	:*letter*	Predefined string. See the predefined regular expressions listed below.

Predefined Regular Expressions

Letter	Meaning	Description
:a	[a–zA–Z0–9]	Alphanumeric
:b	([\t]#)	White space
:c	[a–zA–Z]	Alphabetic
:d	[0–9]	Digit
:f	([~ /\\\" \ [\ ] \ : <|>+=; ,.]#!..!.)	Portion of a file name
:h	([0–9a–fA–F]#)	Hexadecimal number
:i	([a–zA–Z_$] [a–zA–Z0–9_$]@)	C-language identifier
:n	([0–9]#.[0–9]@! [0–9]@.[0–9]#![0–9]#)	Number
:p	(([a–z] \ :!)(\ \ !)(:f(.:f !)\ \) @:f (.:f !))	Path
:q	("[~"]@"! ' '[~ ']@ ')	Quoted string
:w	([a–zA–Z]#)	Word
:z	([0–9]#)	Integer

Summary

QuickHelp is a stand-alone utility that displays help files. Specify the options in the QH environment variable or on the command line. When QuickHelp starts, it first processes all options in the QH environment. It then processes any additional options on the command line.

Syntax

QH [[*options*]] *topic*

Options

/d *filename*	Specifies either a specific database name or a path where the databases are found. If *filename* is specified, then that database is loaded. If a path is specified, all files with the extension .HLP are loaded. Instead of a path, you can specify an environment variable by preceding it with a dollar sign and following it with a colon: for example, /d $INCLUDE:*.HLP.
/l*number*	Specifies the number of lines the QuickHelp window should occupy. If you specify more lines than the current screen mode allows, QuickHelp uses the maximum number allowed by the current screen mode.
/m*number*	Changes the screen mode to display the specified number of lines, where *number* is in the range 25-50.
/p *filename*	Sets the name of the paste file. This option must be followed by a fully qualified file name. This option acts in the same way as the Rename Paste File command on the File menu. The default paste file is PASTE.QH, and the file is placed in the TMP directory.
/pa [[*filename*]]	Specifies that pasting operations are to be appended to the current paste file (rather than overwriting the file). You can follow this option with the name of a file if you do not want paste operations to go to the default file PASTE.QH in the TMP directory.
/q	Prevents the version box from being displayed when QuickHelp is installed as a keyboard monitor.

/r command

Specifies the command that QuickHelp should execute when the Right mouse button is pressed. The default action is to simulate double-clicking the Left mouse button. In other words, single-clicking the Right mouse button is identical to double-clicking the Left mouse button. The following commands are available to change this behavior:

Command	Meaning
l	Pressing the Right mouse button will display the last topic viewed. This is identical to the View Last command on the View menu.
i	Pressing the Right mouse button will display a history of the last topics viewed. This is identical to the View History command on the View menu.
w	Pressing the Right mouse button will temporarily hide the QuickHelp window, allowing you to select a topic from the screen that was displayed prior to activating QuickHelp. This is identical to the Hide Window command on the View menu.
b	Pressing the Right mouse button will display the historically previous topic. This is identical to pressing ALT+F1.
e	Pressing the Right mouse button will continue the search for a topic—displaying the next topic found, if there are any additional topics with the same name. This is identical to the Continue Search command on the View menu.
t	Pressing the Right mouse button will display the table of contents for the current topic. This is identical to the Contents command on the View menu.

/s

Specifies that clicking the mouse above or below the scroll box causes QuickHelp to scroll by lines rather than by pages.

/sgnumber

Specifies the number of screen groups that QuickHelp should monitor, where *number* is in the range 1-12. This option is valid only when QuickHelp is detached from an OS/2 protected-mode screen group. The default is to monitor the first six screen groups created.

/t name

Directs QuickHelp to copy the specified section of the given topic to the current paste file. This option must be followed by a section name. If the paste-file mode is Append (/pa), QuickHelp displays the specified topic in the window. If the paste-file mode is Overwrite (/p), QuickHelp exits immediately after copying the section to the paste file. The following lists the possible topic sections:

Topic	Description
All	Specifies the entire topic
Syntax	Specifies the syntax section of the topic
Example	Specifies the example of the topic

When this option is specified, QuickHelp does not display its window. Instead, it searches for the topic specified on the command line, pastes the topic to the paste file, and exits. This command is normally used only with DOS editors that spawn a shell to QuickHelp to get the paste file. This option is ignored if QuickHelp is installed as a keyboard monitor.

/u

Specifies that QuickHelp is being run by a utility. If the topic specified on the command line is not found, QuickHelp immediately exits with an exit mode of 3.

Summary

RM moves a file to the DELETED directory. You can use the UNDEL command to recover the file.

Syntax

RM [[*options*]] *files*

Options

Options can be grouped.

/f	Deletes read-only files without prompting.
/help	Displays the help file you are reading.
/i	Inquires for user permission before removing.
/k	Keeps read-only files without prompting.
/r	Recurses into subdirectories. A subdirectory to recurse into must be specified. If one is not specified, RM interprets the file name as the name of the subdirectory.

Summary

UNDEL undeletes a deleted file by moving it from the DELETED subdirectory to the parent directory.

Syntax

UNDEL ||**/HELP**|| {[*filename*]|

The /HELP option causes UNDEL to attempt to call the QuickHelp program.

If *filename* is not given, the contents of the DELETED subdirectory are listed. If there is more than one version of the file, you are given a list to choose from. If another copy of the deleted file exists in the parent directory, the current version is moved to the DELETED directory and the previously deleted version is moved to the parent directory.

PART 2

Language Reference

Keywords 99

Statements 99

Escape Sequences 100

Operators 101

Data Type Sizes 102

Preprocessor Directives 103

Preprocessor Operators 104

Pragmas 105

Keywords

Keywords preceded by an underscore (_) are specific to Microsoft C.

_asm	**do**	**for**	**_pascal**	**static**
auto	**double**	**_fortran**	**register**	**struct**
_based	**else**	**goto**	**return**	**switch**
break	**_emit**	**_huge**	**_saveregs**	**typedef**
case	**enum**	**if**	**_segment**	**union**
_cdecl	**_export**	**int**	**_segname**	**unsigned**
char	**extern**	**_interrupt**	**_self**	**void**
const	**_far**	**_loadds**	**short**	**volatile**
continue	**_fastcall**	**long**	**signed**	**while**
default	**float**	**_near**	**sizeof**	

Statements

A C statement consists of program-control keywords, expressions, and function calls. All single statements end with a semicolon. In the following list, *statement* can be either a single statement ending with a semicolon or a block of statements enclosed in braces.

Statement	Action
break;	Terminates the smallest enclosing **do**, **for**, **switch**, or **while** statement in which it appears.
continue;	Passes control to the next iteration of the smallest enclosing **do**, **for**, or **while** statement in which it appears.
do *statement* **while**(*expression*);	Executes *statement* repeatedly until *expression* becomes false (0). The *statement* always executes at least once because the exit condition is tested at the end of the loop.
for([[*initexpr*]]; [[*condition*]]; [[*loopexpr*]]) *statement*	Executes *statement* repeatedly. First the initializing expression *initexpr* is evaluated (separate multiple initializing expressions with commas). Then while *condition* evaluates to a nonzero value, *statement* is executed and *loopexpr* is evaluated. When *condition* becomes 0, control passes to the statement following the **for** loop.
goto *name*; . . . *name*: *statement*	Transfers control directly to the statement specified by the label *name*. The label *name* must be in the same function as the **goto** statement.

if(*expression*)
  statement1
[[**else**
  statement2]]
	Executes *statement1* if *expression* is true (nonzero). If *expression* is false and an **else** is present, control passes to *statement2*.

;	Empty statement; has no effect.
return [[*expression*]];	Terminates execution of the function in which it appears and returns control and the value of *expression* to the calling function.

switch(*expression*)
{
[[**case** *constexpr*:]]
  .
  .
  .
  [[*statement*]]
  .
  .
  .
  [[**default** :
  statement]]
}
	Evaluates *expression* and executes any statement associated with the constant expression *constexpr* whose value matches the initial expression. All statements following **case** are executed until a **break** is encountered or the **switch** block ends. If there is no match with a constant expression, the statement associated with the **default** keyword is executed. If the **default** keyword is not used, control passes to the statement following the **switch** block.

while(*expression*) *statement*	Executes *statement* repeatedly until *expression* becomes 0. The exit condition is tested on entry. If *expression* is initially false, *statement* never executes.

Escape Sequences

\a	Alert (bell)
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

\'	Single quotation mark
\"	Double quotation mark
\\	Backslash
\0*ddd*	ASCII character in octal notation
\x*dd*	ASCII character in hexadecimal notation

Operators

C operators have two properties: precedence and associativity. Within a given level, all operators have the same precedence.

	Operator	Name or Meaning	Associativity
Highest Precedence	( )	Function call	Left to right
	[ ]	Array element	
	.	Member selection (structure or union)	
	–>	Member selection (pointer to structure)	
	++	Increment	Right to left
	– –	Decrement	
	:>	Base operator	Left to right
	!	Logical NOT	Right to left
	~	Bitwise complement	
	–	Arithmetic negation	
	+	Unary plus	
	&	Address	
	*	Indirection	
	sizeof	Size in bytes	
	(*type*)	Type cast	
	*	Multiplication	Left to right
	/	Division	
	%	Remainder	
	+	Addition	Left to right
	–	Subtraction	
	<<	Left shift	Left to right
	>>	Right shift	

Operator	Name or Meaning	Associativity
<	Less than	Left to right
<=	Less than or equal to	
>	Greater than	
>=	Greater than or equal to	
==	Equality	Left to right
!=	Inequality	
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise inclusive OR	Left to right
&&	Logical AND	Left to right
||	Logical OR	Left to right
e1?e2:e3	Conditional	Right to left
=	Simple assignment	Right to left
*=	Multiplication assignment	
/=	Division assignment	
%=	Modulus assignment	
+=	Addition assignment	
-=	Subtraction assignment	
<<=	Left-shift assignment	
>>=	Right-shift assignment	
&=	Bitwise-AND assignment	
^=	Bitwise-exclusive-OR assignment	
|=	Bitwise-inclusive-OR assignment	
,	Comma	Left to right

Lowest Precedence

Data Type Sizes

The /J command-line option makes the **char** default to **unsigned** instead of **signed**. Any integral type that can be modified by the **unsigned** keyword can also be modified by **signed**, which is most useful when the /J option is in effect.

Type Name	Alternative Names	Storage	Range of Values
char	**signed char**	1 byte	–128 to 127
unsigned char	—	1 byte	0 to 255
int	**signed**, **signed int**	2 bytes	32,768 to 32,767
short	**short int**, **signed short**, **signed short int**	2 bytes	–32,768 to 32,767
unsigned short	**unsigned short int**	2 bytes	0 to 65,535
long	**long int**, **signed long**, **signed long int**	4 bytes	2,147,483,648 to 2,147,483,647
unsigned long	**unsigned long int**	4 bytes	0 to 4,294,967,295
_segment		2 bytes	0 to 65,535
enum	—	2 bytes	–32,768 to 32,767
float	—	4 bytes	Approximately 1.2E–38 to 3.4E+38 (7-digit precision)
double	—	8 bytes	Approximately 2.2E–308 to 1.8E+308 (15-digit precision)
long double	—	10 bytes	Approximately 1.2E+4932 to 3.4E–4932 (19-digit precision)

Preprocessor Directives

#define *id sub*
#define *id*(*param*) *sub*

Defines a text-substitution macro or a function macro. All subsequent cases of the identifier *id* are replaced by the substitution text *sub*. If a parenthesized parameter list *param* follows *id*, each occurrence of *id*(*param*) is replaced with a version of *sub* modified by substituting actual arguments for formal parameters.

#undef *identifier*	Removes (undefines) the current definition of *identifier*.
#error *message*	Displays a diagnostic message on the standard error and returns a nonzero (error) code when the compiler terminates.
#include *"path-spec"* **#include** <*path-spec*>	Treats the contents of the file specified by *path-spec* as if it were part of the current file. If *path-spec* is enclosed in double quotation marks, the current directory is searched first, followed by any other directories in the path. If *path-spec* is enclosed in angle brackets, the standard include directory (defined by the environment variable INCLUDE) is searched first.
#if *expression* [[*text*]] [[**#elif** *expression* *text*]] . . . [[**#else** *text*]] **#endif**	Checks each *expression* associated with a **#if** or **#elif** directive until a nonzero restricted constant expression is found, then processes the *text* associated with that restricted constant expression. If there is a **#else** clause, the *text* associated with it is only processed if no restricted constant expression in the **#if** or **#elif** clauses has a nonzero value.
#ifdef *id* **#if defined** (*id*) . . . **#endif**	Checks to see whether *id* is currently defined. The **#ifdef** directive processes the statements within the block if *id* is currently defined, or skips ahead to the **#endif** if it isn't. The **#ifndef** directive processes the statements if the definition is missing, or skips ahead if it is present. The **#ifdef** directive is equivalent to the ANSI-standard directive **#if defined**(*id*), and **#ifndef** is equivalent to **#if !defined**(*id*).
#line *constant* [["*name* "]]	Changes the compiler's internally-stored line number to *constant*, and changes the internal file name to *filename*. The current line number and file name are available through the predefined identifiers **__LINE__** and **__FILE__**.
#pragma *directive*	Instructs the compiler to effect the implementation-specific feature indicated by *directive*.

Preprocessor Operators

#operand	Puts quotation marks around *operand*, treating it as a string
token1##*token2*	Pastes *token1* and *token2* together
#@*parameter*	Expands the argument to a quoted character

Pragmas

alloc_text — **#pragma alloc_text** (*text-segment*, *function1*[[, *function2*]]...)

Places the specified functions in the given text segment.

check_pointer — **#pragma check_pointer** ([[{ on | off }]])

Instructs the compiler to turn off pointer checking if off is specified, or turn on pointer checking if on is specified. The **check _pointer** pragma only works when you compile with the Quick Compile (/qc) option. If **check _pointer** is specified with no arguments, it toggles the pointer-checking setting currently in effect.

check_stack — **#pragma check_stack** ([[{ on | off }]])

Instructs the compiler to turn off stack probes if off is given, or turn on stack probes if on is given. If no argument is given, stack probes are treated according to the default (on unless /Gs was used).

comment — **#pragma comment(** *commenttype* [[,*"commentstring"*]]...)

Directs the compiler to place a comment record in the object file. The parameter *commenttype* specifies a **compiler**, **exestr**, **lib**, or **user** comment record, and *commentstring* (enclosed in quotation marks) provides additional information.

function — **#pragma function(** *function1*[[, *function2...*]])

Directs the compiler to generate function calls instead of the intrinsic forms for each function listed.

intrinsic — **#pragma intrinsic(** *function1*[[, *function2...*]])

Instructs the compiler to generate the intrinsic forms of the listed functions.

linesize — **#pragma linesize(** [[*characters*]])

Sets the number of characters per line in the source listing.

loop_opt — **#pragma loop_opt(** [[{ off | on }]])

Turns loop optimizing on or off. Without an argument, the pragma directs the compiler to revert to the behavior specified on the command line.

message **#pragma message(** *messagestring* **)**

Sends a message to the standard output. The *messagestring* parameter is a string literal that contains the message.

optimize **#pragma optimize(** "[*o_string*]" { off | on } **)**

Enables or disables one or more optimizing options. The optional *o_string* list can include one or more of the following characters:

a	Assume no aliases in the following code
c	Enable local common subexpressions
e	Enable global register allocation
g	Enable global common subexpressions
l	Optimize loops
n	Disable unsafe loop optimizations
p	Ensure floating-point consistency
t	Optimize for speed
w	Assume no aliases except across function calls

If the list is empty, the compiler reverts to the default settings. If **on** or **off** is omitted, the given option is toggled.

pack **#pragma pack(** [*number*] **)**

Specifies packing alignment for **struct** types, where *number* is 1, 2, or 4. Without an argument, packing reverts to the behavior specified on the command line. The **/Zp** option without *number* is equivalent to **#pragma pack(1)**. In the absence of the **/Zp** option, the default is **#pragma pack(2)**.

page **#pragma page(** [*pages*] **)**

Directs the compiler to skip the specified number of pages in the source listing where the pragma appears.

pagesize **#pragma pagesize(** [*lines*] **)**

Sets the number of lines per page in the source listing.

same_seg **#pragma same_seg(** *variable1* [, *variable2*]...)

Directs the compiler to assume that the **external far** variables are allocated in the same segment. The /ND option must be used when compiling the modules containing the listed variables to ensure that these variables are actually allocated in the given data segment.

skip **#pragma skip(** [*lines*])

Skips specified number of *lines* and generates a newline in the source listing at the point where the pragma appears.

subtitle **#pragma subtitle(** "*subtitlename*")

Sets a subtitle in the source listing.

title **#pragma title(** "*titlename*")

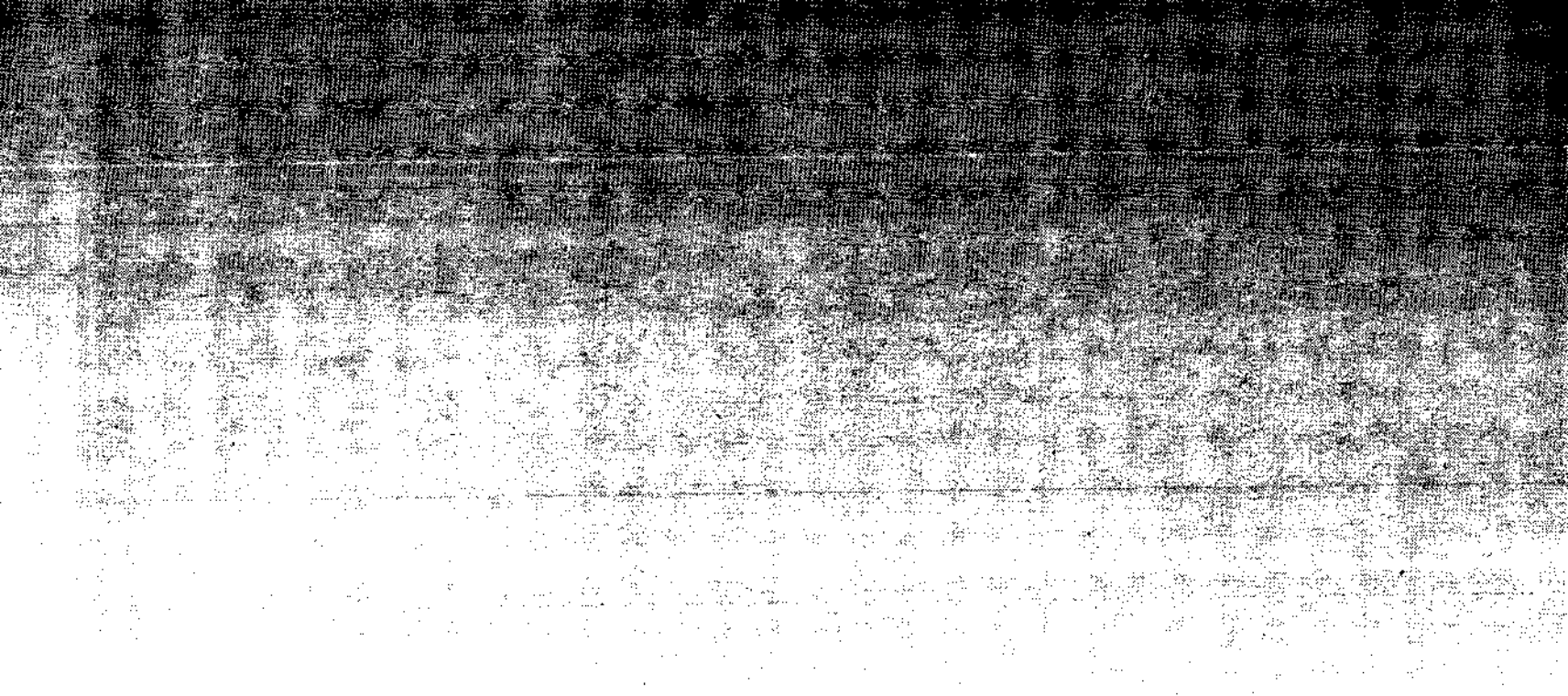
Sets the title for the source listing. It appears in the upper left corner of each page of the listing.

PART 3

Run-Time Library Functions

Routines by Category 111

Alphabetical Listing 135



Routines by Category

The Microsoft C run-time library contains approximately 500 routines useful for performing common programming tasks. In this part of the *C Reference* the routines are listed first by category of tasks and then alphabetically.

The alphabetical listing is more complete. It gives the following information for each routine: the prototype, the header files that contain the routine, a definition of each argument, the return value, and a brief description of the function the routine performs. In addition, check boxes indicate whether the routine is ANSI-compatible and the operating systems it works under.

This section lists and briefly describes the library routines according to the categories of tasks they perform. These are the main categories of library routines:

Buffer manipulation

Character classification and conversion

Data conversion

Directory control

File handling

Graphics input and output

Internationalization

Math

Memory allocation

Process and environment control

Searching and sorting

String manipulation

System calls

Time

Variable-length argument lists

Buffer Manipulation

The buffer-manipulation routines are useful for working with areas of memory on a character-by-character basis. Routines preceded by the _f prefix are model-independent versions.

Routine	Use
memccpy, _fmemccpy	Copies characters from one buffer to another, until a given character or a given number of characters has been copied
memchr, _fmemchr	Returns a pointer to the first occurrence, within a specified number of characters, of a given character in the buffer
memcmp, _fmemcmp	Compares a specified number of characters from two buffers
memcpy, _fmemcpy	Copies a specified number of characters from one buffer to another
memicmp, _fmemicmp	Compares a specified number of characters from two buffers without regard to the case of the letters (uppercase and lowercase treated as equivalent)
memmove, _fmemmove	Copies a specified number of characters from one buffer to another
memset, _fmemset	Uses a given character to initialize a specified number of bytes in the buffer
swab	Swaps bytes of data and stores them at the specified location

Character Classification and Conversion

The character classification and conversion routines let you test individual characters in a variety of ways and convert between uppercase and lowercase characters.

Routine	Use
isalnum	Tests for alphanumeric character
isalpha	Tests for alphabetic character
isascii	Tests for ASCII character
iscntrl	Tests for control character

isdigit	Tests for decimal digit
isgraph	Tests for printable character except space
islower	Tests for lowercase character
isprint	Tests for printable character
ispunct	Tests for punctuation character
isspace	Tests for white-space character
isupper	Tests for uppercase character
isxdigit	Tests for hexadecimal digit
toascii	Converts character to ASCII code
tolower	Tests character and converts to lowercase if uppercase
_tolower	Converts character to lowercase (unconditional)
toupper	Tests character and converts to uppercase if lowercase
_toupper	Converts character to uppercase (unconditional)

Data Conversion

The data-conversion routines convert numbers to strings of ASCII characters, and vice versa. These routines are implemented as functions.

Routine	Use
abs	Finds absolute value of integer
atof	Converts string to **float**
atoi	Converts string to **int**
atol	Converts string to **long**
_atold	Converts string to **long double**
ecvt	Converts **double** to string
fcvt	Converts **double** to string
gcvt	Converts **double** to string
itoa	Converts **int** to string
labs	Finds absolute value of long integer

ltoa	Converts **long** to string
strtod	Converts string to **double**
strtol	Converts string to a **long** integer
_strtold	Converts string to **long double**
strtoul	Converts string to an **unsigned long** integer
ultoa	Converts **unsigned long** to string

Directory Control

These routines let a program access, modify, and obtain information about the directory structure. These routines are functions.

Routine	Use
chdir	Changes current working directory
_chdrive	Changes current drive
getcwd	Gets current working directory
_getdcwd	Gets current working directory for the specified drive
_getdrive	Gets the current disk drive
mkdir	Makes a new directory
rmdir	Removes a directory
_searchenv	Searches for a given file on specified paths

File Handling

The file-handling routines let you create, manipulate, and delete files. They also set and check file-access permissions.

Routine	Use
access	Checks file-permission setting
chmod	Changes file-permission setting
chsize	Changes file size
filelength	Gets file length
fstat	Gets file-status information on handle

_fullpath	Makes an absolute path name from a relative path name
isatty	Checks for character device
locking	Locks areas of file (available with OS/2 and DOS versions 3.0 and later)
_makepath	Merges path-name components into a single, full path name
mktemp	Creates unique file name
remove	Deletes file
rename	Renames file
setmode	Sets file-translation mode
_splitpath	Splits a path name into component pieces
stat	Gets file-status information on named file
umask	Sets default-permission mask
unlink	Deletes file

Graphics

Microsoft C graphics routines offer a wide variety of graphics functions, low-level graphics primitives, font functions, and presentation graphics (displays such as graphs and pie charts).

Low-Level Graphics and Character-Font Functions

Routine	Use
_arc, _arc_w, _arc_wxy	Draws an arc
_clearscreen	Clears the screen and fills it with the current back ground color
_displaycursor	Sets the cursor on or off on exit from a graphics routine
_ellipse, _ellipse_w, _ellipse_wxy	Draws an ellipse or circle
_floodfill, _floodfill_w	Flood-fills an area of the screen with the current color

_getactivepage	Gets the current active page number
_getarcinfo	Determines the end points in viewport coordinates of the most recently drawn arc or pie
_getbkcolor	Returns the current background color
_getcolor	Gets the current color
_getcurrentposition, **_getcurrentposition_w**	Obtains the current graphics-output position used by **_lineto** and **_outgtext**
_getfillmask	Gets the current fill mask
_getfontinfo	Obtains the current font characteristics
_getgtextextent	Determines the width in pixels of specified text in the current font
_getgtextvector	Gets orientation of font text output
_getimage, **_getimage_w**, **_getimage_wxy**	Stores a screen image in memory
_getlinestyle	Gets the current line style mask
_getphyscoord	Converts view coordinates to physical coordinates
_getpixel, **_getpixel_w**	Obtains a pixel's color
_gettextcolor	Obtains the current text color
_gettextcursor	Returns the current cursor attribute (text modes only)
_gettextposition	Obtains the current text-output position
_gettextwindow	Gets the current text window boundaries
_getvideoconfig	Obtains status of current graphics environment
_getvisualpage	Gets the current visual page number
_getviewcoord	Converts physical coordinates to view coordinates
_getviewcoord_w	Converts window coordinates to view coordinates
_getviewcoord_wxy	Converts window coordinates in **_wxycoord** structure to view coordinates
_getwindowcoord	Converts view coordinates to window coordinates
_getwritemode	Gets the current logical write mode for line drawing
_grstatus	Returns the status of the most recent graphics function call

_imagesize, **_imagesize_w**, **_imagesize_wxy**	Returns the size (in bytes) of the buffer needed to store the image
_lineto, **_lineto_w**	Draws a line from the current graphics output position to a specified point
_moveto, **_moveto_w**	Moves the current graphics-output position to a specified point
_outgtext	Outputs the null-terminated text string using the current font and rotation on the screen at the current graphics-output position
_outmem	Prints text of a specified length from a memory buffer
_outtext	Outputs a text string to the screen at the current text position
_pie, **_pie_w**, **_pie_wxy**	Draws a pie-slice-shaped figure
_polygon, **_polygon_w**, **_polygon_wxy**	Draws or scan-fills a polygon
_putimage, **_putimage_w**	Retrieves an image from memory and displays it
_rectangle, **_rectangle_w**, **_rectangle_wxy**	Draws or scan-fills a rectangle
_registerfonts	Initializes the font graphics system
_remapallpalette	Changes all colors in the current palette
_remappalette	Changes a single color in the current palette
_scrolltextwindow	Scrolls the current text window up or down
_selectpalette	Selects a predefined palette
_setactivepage	Sets memory area for the current active page
_setbkcolor	Sets the current background color
_setcliprgn	Limits graphic output to a region of the screen
_setcolor	Sets the current color
_setfillmask	Sets the current fill mask

_setfont	Finds a single font from the set of registered fonts
_setgtextvector	Sets the current orientation for font text output
_setlinestyle	Sets the current line style mask
_setpixel, _setpixel_w	Sets a pixel's color
_settextcolor	Sets the current text color
_settextcursor	Sets the current cursor attribute (text modes only)
_settextposition	Sets the current text position
_settextrows	Sets the number of text rows
_settextwindow	Defines the current text-display window
_setvideomode	Sets the video mode for the display screen
_setvideomoderows	Sets the video mode and the number of text rows
_setvieworg	Positions the view coordinate origin
_setviewport	Limits graphics output to a region of the screen and positions the view coordinate origin to the upper left corner of that region
_setvisualpage	Sets memory area for the current visual page
_setwindow	Defines a floating-point window coordinate system
_setwritemode	Sets logical write mode for line drawing
_unregisterfonts	Frees memory previously allocated and used by **_registerfonts**
_wrapon	Enables or disables line wrap

Presentation Graphics Functions

Routine	Use
_pg_analyzechart	Analyzes a single series of data for a bar, column, or line chart
_pg_analyzechartms	Analyzes a multiple series of data for a bar, column, or line chart
_pg_analyzepie	Analyzes data for a pie chart

_pg_analyzescatter	Analyzes a single series of data for a scatter diagram
_pg_analyzescatterms	Analyzes a multiple series of data for a scatter diagram
_pg_chart	Displays a single-series bar, column, or line chart
_pg_chartms	Displays a multiple-series bar, column, or line chart
_pg_chartpie	Displays a pie chart
_pg_chartscatter	Displays a scatter diagram for a single series of data
_pg_chartscatterms	Displays a scatter diagram for more than one series of data
_pg_defaultchart	Initializes all necessary variables in the chart environment for a specified chart type
_pg_getchardef	Retrieves the current 8-by-8-pixel bit map for a specified character
_pg_getpalette	Retrieves current colors, line styles, fill patterns, and plot characters for all presentation graphics palettes
_pg_getstyleset	Retrieves the contents of the current styleset
_pg_hlabelchart	Writes text horizontally on the screen
_pg_initchart	Initializes the presentation graphics library
_pg_resetpalette	Sets current colors, line styles, fill patterns, and plot characters to the default values for the current screen mode
_pg_resetstyleset	Resets the contents of the current styleset to the default value for the current screen mode
_pg_setchardef	Sets the 8-by-8-pixel bit map for a specified character
_pg_setpalette	Sets current colors, line styles, fill patterns, and plot characters for all presentation graphics palettes
_pg_setstyleset	Sets the contents of the current styleset
_pg_vlabelchart	Writes text vertically on the screen

Input and Output

The input and output (I/O) routines of the standard C library allow you to read and write data to and from files and devices.

Stream Routines

Routine	Use
clearerr	Clears the error indicator for a stream
fclose	Closes a stream
fcloseall	Closes all open streams
fdopen	Associates a stream with an open file handle
feof	Tests for end-of-file on a stream
ferror	Tests for error on a stream
fflush	Flushes a stream
fgetc	Reads a character from a stream (function version)
fgetchar	Reads a character from **stdin** (function version)
fgetpos	Gets the position indicator of a stream
fgets	Reads a string from a stream
fileno	Gets file handle associated with a stream
flushall	Flushes all streams
fopen	Opens a stream
fprintf	Writes formatted data to a stream
fputc	Writes a character to a stream (function version)
fputchar	Writes a character to **stdout** (function version)
fputs	Writes a string to a stream
fread	Reads unformatted data from a stream
freopen	Reassigns a **FILE** pointer to a new file
fscanf	Reads formatted data from a stream
fseek	Moves file position to a given location
fsetpos	Sets the position indicator of a stream

_fsopen	Opens a stream with file sharing
ftell	Gets current file position
fwrite	Writes unformatted data items to a stream
getc	Reads a character from a stream
getchar	Reads a character from **stdin**
gets	Reads a line from **stdin**
getw	Reads a binary **int** item from stream
printf	Writes formatted data to **stdout**
putc	Writes a character to a stream
putchar	Writes a character to **stdout**
puts	Writes a line to a stream
putw	Writes a binary **int** item to a stream
rewind	Moves file position to beginning of a stream
rmtmp	Removes temporary files created by **tmpfile**
scanf	Reads formatted data from **stdin**
setbuf	Controls stream buffering
setvbuf	Controls stream buffering and buffer size
sprintf	Writes formatted data to string
sscanf	Reads formatted data from string
tempnam	Generates a temporary file name in given directory
tmpfile	Creates a temporary file
tmpnam	Generates a temporary file name
ungetc	Places a character in the buffer
vfprintf	Writes formatted data to a stream
vprintf	Writes formatted data to **stdout**
vsprintf	Writes formatted data to a string

Low-Level Routines

Routine	Use
close	Closes a file
creat	Creates a file
dup	Creates a second handle for a file
dup2	Reassigns a handle to a file
eof	Tests for end-of-file
lseek	Repositions file pointer to a given location
open	Opens a file
read	Reads data from a file
sopen	Opens a file for file sharing
tell	Gets current file-pointer position
umask	Sets default file-permission mask
write	Writes data to a file

Console and Port I/O

Routine	Use
cgets	Reads a string from the console
cprintf	Writes formatted data to the console
cputs	Writes a string to the console
cscanf	Reads formatted data from the console
getch	Reads a character from the console
getche	Reads a character from the console and echoes it
inp	Reads one byte from the specified I/O port
inpw	Reads a two-byte word from the specified I/O port
kbhit	Checks for a keystroke at the console

outp	Writes one byte to the specified I/O port
outpw	Writes a two-byte word to the specified I/O port
putch	Writes a character to the console
ungetch	"Ungets" the last character read from the console so that it becomes the next character read

Internationalization

Internationalization routines are useful for creating different versions of a program for international markets.

Routine	Use
localeconv	Sets a structure with appropriate values for formatting numeric quantities
setlocale	Selects the appropriate locale for the program
strcoll	Compares strings using locale-specific information
strftime	Formats a date and time string
strxfrm	Transforms a string based on locale-specific information

Math

The math routines allow you to perform common mathematical calculations. Many math routines work with floating-point values and therefore require floating-point support, which is linked in from the C run-time library. Routines ending with **l** support an 80-bit data type for the corresponding routines. These routines return a **long double** value.

Routine	Use
acos, acosl	Calculates the arccosine
asin, asinl	Calculates the arcsine
atan, atanl	Calculates the arctangent
atan2, atan2l	Calculates the arctangent
bessel[1]	Calculates Bessel functions
cabs, cabsl	Finds the absolute value of a complex number

ceil, ceill	Finds the integer ceiling
_clear87	Gets and clears the floating-point status word
_control87	Gets the old floating-point control word and sets a new control-word value
cos, cosl	Calculates the cosine
cosh, coshl	Calculates the hyperbolic cosine
dieeetomsbin	Converts IEEE double-precision number to Microsoft (MS) binary format
div	Divides one integer by another, returning the quotient and remainder
dmsbintoieee	Converts Microsoft binary double-precision number to IEEE format
exp, expl	Calculates the exponential function
fabs, fabsl	Finds the absolute value
fieeetomsbin	Converts IEEE single-precision number to Microsoft binary format
floor, floorl	Finds the largest integer less than or equal to the argument
fmod, fmodl	Finds the floating-point remainder
fmsbintoieee	Converts Microsoft binary single-precision number to IEEE format
_fpreset	Reinitializes the floating-point-math package
frexp, frexpl	Calculates an exponential value
hypot, hypotl	Calculates the hypotenuse of right triangle
ldexp, ldexpl	Calculates the argument times 2^{exp}
ldiv	Divides one **long** integer by another, returning the quotient and remainder
log, logl	Calculates the natural logarithm
log10, log10l	Calculates the base-10 logarithm
_lrotl, _lrotr	Shifts an **unsigned long int** item left (**_lrotl**) or right (**_lrotr**)
matherr, _matherrl	Handles math errors
max, min	Returns the larger or smaller of two values

modf, modfl	Breaks down the argument into integer and fractional parts
pow, powl	Calculates a value raised to a power
rand	Gets a pseudorandom number
_rotl, _rotr	Shifts an **unsigned int** item left (**_rotl**) or right (**_rotr**)
sin, sinl	Calculates the sine
sinh, sinhl	Calculates the hyperbolic sine
sqrt, sqrtl	Finds the square root
srand	Initializes a pseudorandom series
_status87	Gets the floating-point status word
tan, tanl	Calculates the tangent
tanh, tanhl	Calculates the hyperbolic tangent

[1] The **bessel** routine does not correspond to a single function, but to 12 functions named **j0**, **j1**, **jn**, **y0**, **y1**, **yn**, **_j0l**, **_j1l**, **_jnl**, **_y0l**, **_y1l**, and **_ynl**.

Memory Allocation

The memory-allocation routines allow you to allocate, free, and reallocate blocks of memory.

Routine	Use
alloca	Allocates a block of memory from the program's stack
_bfreeseg	Frees a based heap
_bheapseg	Allocates a based heap
calloc, _bcalloc, _fcalloc, _ncalloc	Allocates storage for an array
_expand, _bexpand, _fexpand, _nexpand	Expands or shrinks a block of memory without moving its location
free, _bfree, _ffree, _nfree	Frees an allocated block
_freect	Returns approximate number of items of given size that could be allocated in the near heap

halloc	Allocates storage for huge array
_heapadd, _bheapadd	Adds memory to a heap
_heapchk, _bheapchk, _fheapchk, _nheapchk	Checks a heap for consistency
_heapmin, _bheapmin, _fheapmin, _nheapmin	Releases unused memory in a heap
_heapset, _bheapset, _fheapset, _nheapset	Fills free heap entries with a specified value
_heapwalk, _bheapwalk, _fheapwalk, _nheapwalk	Returns information about each entry in a heap
hfree	Frees a block allocated by **halloc**
malloc, _bmalloc, _fmalloc, _nmalloc	Allocates a block of memory
_memavl	Returns approximate number of bytes available for allocation in the near heap
_memmax	Returns size of largest contiguous free block in the near heap
_msize, _bmsize, _fmsize, _nmsize	Returns size of an allocated block
realloc, _brealloc, _frealloc, _nrealloc	Reallocates a block to a new size
stackavail	Returns size of stack space available for allocation with **alloca**

Process and Environment Control

The process-control routines allow you to start, stop, and manage processes from within a program. Environment-control routines allow you to get and change information about the operating-system environment.

Routine	Use
abort	Aborts a process without flushing buffers or calling functions registered by **atexit** and **onexit**
assert	Tests for logic error
atexit	Schedules routines for execution at program termination
_beginthread	Creates an execution thread (OS/2 only)
_cexit	Performs the **exit** termination procedures (such as flushing buffers) and returns control to the calling program
_c_exit	Performs the **_exit** termination procedures without performing clean-up operations and returns control to the calling program
cwait	Suspends the calling process until a specified child process terminates (OS/2 only)
_endthread	Terminates an execution thread (OS/2 only)
execl	Executes child process with argument list
execle	Executes child process with argument list and given environment
execlp	Executes child process using PATH variable and argument list
execlpe	Executes child process using PATH variable, given environment, and argument list
execv	Executes child process with argument array
execve	Executes child process with argument array and given environment
execvp	Executes child process using PATH variable and argument array
execvpe	Executes child process using PATH variable, given environment, and argument array
exit	Calls functions registered by **atexit** and **onexit**, then flushes all buffers and closes all open files before terminating the process
_exit	Terminates process without processing **atexit** or **onexit** functions or flushing buffers

getenv	Gets the value of an environment variable
getpid	Gets process ID number
longjmp	Restores a saved stack environment
onexit	Schedules routines for execution at program termination
_pclose	Waits for a child command and closes the pipe on the associated stream
perror	Prints error message
_pipe	Creates a pipe
_popen	Creates a pipe and asynchronously executes a child copy of the command processor
putenv	Adds or changes the value of an environment variable
raise	Sends a signal to the calling process
setjmp	Saves a stack environment
signal	Handles an interrupt signal
spawnl	Executes child process with argument list
spawnle	Executes child process with argument list and given environment
spawnlp	Executes child process using PATH variable and argument list
spawnlpe	Executes child process using PATH variable, given environment, and argument list
spawnv	Executes child process with argument array
spawnve	Executes child process with argument array and given environment
spawnvp	Executes child process using PATH variable and argument array
spawnvpe	Executes child process using PATH variable, given environment, and argument array
system	Executes an operating-system command
wait	Suspends the calling process until any of the caller's immediate child processes terminate (OS/2 only)

Searching and Sorting

Search and sort routines provide binary-search, linear-search, and quick-sort capabilities.

Routine	Use
bsearch	Performs binary search
lfind	Performs linear search for given value
lsearch	Performs linear search for given value, which is added to array if not found
qsort	Performs quick sort

String Manipulation

The string functions allow you to compare strings, copy them, search for strings and characters, and perform various other operations.

Routine	Use
strcat, _fstrcat	Appends one string to another
strchr, _fstrchr	Finds first occurrence of a given character in string
strcmp, _fstrcmp	Compares two strings
strcpy, _fstrcpy	Copies one string to another
strcspn, _fstrcspn	Finds first occurrence of a character from given character set in string
strdup, _fstrdup, _nstrdup	Duplicates a string
strerror	Maps an error number to a message
_strerror	Maps a user-defined error message to a string
stricmp, _fstricmp	Compares two strings without regard to case
strlen, _fstrlen	Finds length of string
strlwr, _fstrlwr	Converts string to lowercase
strncat, _fstrncat	Appends characters of string
strncmp, _fstrncmp	Compares characters of two strings
strncpy, _fstrncpy	Copies characters of one string to another

strnicmp, _fstrnicmp	Compares characters of two strings without regard to case
strnset, _fstrnset	Sets characters of string to given character
strpbrk, _fstrpbrk	Finds first occurrence of character from one string in another
strrchr, _fstrrchr	Finds last occurrence of given character in string
strrev, _fstrrev	Reverses string
strset, _fstrset	Sets all characters of string to given character
strspn, _fstrspn	Finds first substring from given character set in string
strstr, _fstrstr	Finds first occurrence of given string in another string
strtok, _fstrtok	Finds next token in string
strupr, _fstrupr	Converts string to uppercase

System Calls

The following routines give access to IBM-PC BIOS (Basic Input/Output System) interrupts and DOS system calls.

BIOS Interface

Routine	Use
_bios_disk	Issues service requests for both hard and floppy disks, using INT 0x13
_bios_equiplist	Performs an equipment check, using INT 0x11
_bios_keybrd	Provides access to keyboard services, using INT 0x16
_bios_memsize	Obtains information about available memory, using INT 0x12
_bios_printer	Performs printer output services, using INT 0x17
_bios_serialcom	Performs serial communications tasks, using INT 0x14
_bios_timeofday	Provides access to system clock, using INT 0x1A

DOS Interface

Routine	Use
bdos	Invokes DOS system call; uses only DX and AL registers
_chain_intr	Chains one interrupt handler to another
_disable	Disables interrupts
_dos_allocmem	Allocates a block of memory, using DOS system call 0x48
_dos_close	Closes a file, using DOS system call 0x3E
_dos_creat	Creates a new file and erases any existing file having the same name, using DOS system call 0x3C
_dos_creatnew	Creates a new file and returns an error if a file having the same name exists, using DOS system call 0x5B
_dos_findfirst	Finds first occurrence of a given file, using DOS system call 0x4E
_dos_findnext	Finds subsequent occurrences of a given file, using DOS system call 0x4F
_dos_freemem	Frees a block of memory, using DOS system call 0x49
_dos_getdate	Gets the system date, using DOS system call 0x2A
_dos_getdiskfree	Gets information on a disk volume, using DOS system call 0x36
_dos_getdrive	Gets the current default drive, using DOS system call 0x19
_dos_getfileattr	Gets current attributes of a file or directory, using DOS system call 0x43
_dos_getftime	Gets the date and time a file was last written, using DOS system call 0x57
_dos_gettime	Gets the current system time, using DOS system call 0x2C
_dos_getvect	Gets the current value of a specified interrupt vector, using DOS system call 0x35

_dos_keep	Installs terminate-and-stay-resident (TSR) programs using DOS system call 0x31
_dos_open	Opens an existing file, using DOS system call 0x3D
_dos_read	Reads a file, using DOS system call 0x3F
_dos_setblock	Changes the size of a previously allocated block, using DOS system call 0x4A
_dos_setdate	Sets the current system date, using DOS system call 0x2B
_dos_setdrive	Sets the default disk drive, using DOS system call 0x0E
_dos_setfileattr	Sets the current attributes of a file, using DOS system call 0x43
_dos_setftime	Sets the date and time that the specified file was last written, using DOS system call 0x57
_dos_settime	Sets the system time, using DOS system call 0x2D
_dos_setvect	Sets a new value for the specified interrupt vector, using DOS system call 0x25
_dos_write	Sends output to a file, using DOS system call 0x40
dosexterr	Obtains in-depth error information from DOS system call 0x59
_enable	Enables interrupts
FP_OFF	Returns offset portion of a far pointer (OS/2 and DOS)
FP_SEG	Returns segment portion of a far pointer (OS/2 and DOS)
_harderr	Establishes a hardware error handler
_hardresume	Returns to DOS after a hardware error
_hardretn	Returns to the application after a hardware error

int86	Invokes DOS interrupts
int86x	Invokes DOS interrupts with segment register values
intdos	Invokes DOS system call using registers other than DX and AL
intdosx	Invokes DOS system call using registers other than DX and AL with segment register values
segread	Returns current values of segment registers (OS/2 and DOS)

Time

The time functions allow you to obtain the current time, then convert and store it. The current time is always taken from the system time.

Routine	Use
asctime	Converts time from type **struct tm** to character string
clock	Returns the elapsed CPU time for a process
ctime	Converts time from long integer to character string
difftime	Computes the difference between two times
ftime	Puts current system time in variable of type **struct tm**
gmtime	Converts time from integer to **struct tm**
localtime	Converts time from integer to **struct tm** with local correction
mktime	Converts time to a calendar value
_strdate	Returns the current system date as a string
strftime	Formats a date and time string
_strtime	Returns the current system time as a string
time	Gets current system time as long integer
tzset	Sets external time variables from environment time variable
utime	Sets file-modification time

Variable-Length Argument Lists

The **va_arg**, **va_end**, and **va_start** routines are macros that provide a portable way to access the arguments to a function when the function takes a variable number of arguments.

Routine	Use
va_arg	Retrieves argument from list
va_end	Resets pointer
va_start	Sets pointer to beginning of argument list

abort

void abort(void);

Include	PROCESS.H or STDLIB.H
Returns	No return value

Aborts the current process.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

abs

int abs(int *n* **);**

Include	STDLIB.H or MATH.H
n	Value whose absolute value is sought
Returns	The absolute value of *n*

Calculates the absolute value of its argument.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

access

int access(char **pathname*, **int** *mode* **);**

Include	IO.H, ERRNO.H
pathname	File or directory path name
mode	**00** (exist), **02** (write), **04** (read), **06** (read/write)
Returns	0 if the file has the given access mode or if the directory exists; –1 if the directory or file does not exist or does not have the given mode
errno	**EACCESS, ENOENT**

Determines whether a specified file or directory exists, and (in the case of a file) whether it can be accessed in the specified mode.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

acos

double acos(double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arccosine is sought
Returns	0 if *x* is less than –1 or greater than 1
errno	**EDOM**

Returns the arccosine of *x* in the range 0 to pi radians.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

acosl

long double acosl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arccosine is sought
Returns	0 if *x* is less than –1 or greater than 1
errno	**EDOM**

Returns the arccosine of *x* in the range 0 to pi radians. Long double-precision floating-point version of **acos**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

alloca

void *alloca(size_t *size* **);**

Include	MALLOC.H
size	Bytes to be allocated from the stack
Returns	A **void** pointer to the allocated space; **NULL** if the space cannot be allocated

Allocates *size* bytes from the program's stack.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_arc

short _far _arc(short *x1*, **short** *y1*, **short** *x2*, **short** *y2*, **short** *x3*, **short** *y3*, **short** *x4*, **short** *y4*);

Include	GRAPH.H
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
x3, *y3*	Start vector
x4, *y4*	End vector
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws an elliptical arc using the view coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_arc_w

short _far _arc_w(double *x1*, **double** *y1*, **double** *x2*, **double** *y2*, **double** *x3*, **double** *y3*, **double** *x4*, **double** *y4*);

Include	GRAPH.H
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
x3, *y3*	Second point of start vector (center of bounding rectangle is first point)
x4, *y4*	Second point of end vector (center of bounding rectangle is first point)
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws an elliptical arc using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_arc_wxy

short _far _arc_wxy(struct _wxycoord _far *_pwxy1_, struct _wxycoord _far *_pwxy2_,
struct _wxycoord _far *_pwxy3_, struct _wxycoord _far *_pwxy4_);

Include	GRAPH.H
pwxy1	Upper left corner of bounding rectangle
pwxy2	Lower right corner of bounding rectangle
pwxy3	Start vector
pwxy4	End vector
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws an elliptical arc using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

asctime

char *asctime(const struct tm *_timeptr_);

Include	TIME.H
timeptr	Time/date structure defined in TIME.H
Returns	A pointer to the character string result

Converts a value stored as a **tm** time structure to a character string.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

asin

double asin(double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arcsine is sought
Returns	0 if *x* is less than –1 or greater than 1
errno	**EDOM**

Calculates the arcsine of a value in the range –pi/2 to pi/2 radians.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

asinl

long double asinl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arcsine is sought
Returns	0 if *x* is less than –1 or greater than 1
errno	**EDOM**

Calculates the arcsine of a value in the range –pi/2 to pi/2 radians. Long double-precision floating-point version of **asin**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

assert

void assert(int *expression* **);**

Include	ASSERT.H, STDIO.H
expression	A logical expression
Returns	No return value

Prints a diagnostic message and calls the **abort** routine if a logical expression is false (0). Activated only if **NDEBUG** is defined at compile time.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

atan

double atan(double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arctangent is sought
Returns	The arctangent result; 0 if *x* is 0
errno	**EDOM**

Calculates the arctangent of *x* in the range –pi/2 to pi/2.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

atanl

long double atanl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arctangent is sought
Returns	The arctangent result; 0 if *x* is 0
errno	**EDOM**

Calculates the arctangent of *x* in the range –pi/2 to pi/2. Long double-precision floating-point version of **atan**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

atan2

double atan2(double *y*, **double** *x* **);**

Include	MATH.H, ERRNO.H
y/x	Value whose arctangent is sought
Returns	The arctangent result; 0 if *x* or *y* is 0
errno	**EDOM**

Calculates the arctangent of *y/x* in the range –pi to pi.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

atan2l

long double atan2l(long double *y*, **long double** *x* **);**

Include	MATH.H, ERRNO.H
y/x	Value whose arctangent is sought
Returns	The arctangent result; 0 if *x* or *y* is 0
errno	**EDOM**

Calculates the arctangent of *y/x* in the range –pi to pi. Long double-precision floating-point version of **atan2**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

atexit

int atexit(void(*******func** **)(void));**

Include	STDLIB.H
func	Function to call on exit
Returns	0 if successful; a nonzero value if not

Calls the specified function at exit. Successive calls create a register of up to 32 functions to be called (last in, first out).

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

atof

double atof(const char *_string_);

Include	MATH.H or STDLIB.H
string	String to be converted
Returns	The converted string; 0 if the string cannot be converted

Converts a character string to a double-precision floating-point value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

atoi

int atoi(const char *_string_);

Include	STDLIB.H
string	String to be converted
Returns	The converted string; 0 if the string cannot be converted

Converts a character string to an integer value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

atol

long atol(const char *_string_);

Include	STDLIB.H
string	String to be converted
Returns	The converted string; 0 if the string cannot be converted

Converts a character string to a long integer value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_atold

long double _atold(const char *_string_);

Include	MATH.H or STDLIB.H
string	String to be converted
Returns	The converted string; 0 if the string cannot be converted

Converts a character string to a long double-precision floating-point value.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bcalloc

void _based(void) *_bcalloc(_segment *seg*, **size_t** *num*, **size_t** *size* **);**

Include	MALLOC.H
seg	Based-heap segment selector
num	Number of elements
size	Length in bytes of each element
Returns	A based pointer to the allocated space; **_NULLOFF** if there is insufficient memory available, if *num* or *size* is 0, or if the specified segment has not been initialized as part of the based heap

Allocates array storage on the specified based heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

bdos

int bdos(int *dosfunc*, **unsigned int** *dosdx*, **unsigned int** *dosal* **);**

Include	DOS.H
dosfunc	Function number
dosdx	DX register value
dosal	AL register value
Returns	The value of the AX register after the system call has completed

Invokes the DOS system call specified by a function number.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_beginthread

int _far _beginthread(void(_far * *start_address* **)(void _far *),**
void _far * *stack_bottom*, **unsigned** *stack_size*, **void _far *** *arglist* **);**

Include	PROCESS.H, STDDEF.H, ERRNO.H
start_address	Address where execution begins
stack_bottom	Address of the thread stack or **NULL**
stack_size	Stack space to reserve
arglist	Address of data item to pass to new thread
Returns	The thread identification number of the new thread if successful; −1 to indicate an error if not
errno	**EINVAL, EAGAIN**

Creates a thread that begins execution at *start_address*.

☐ ANSI ☐ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bexpand

void _based(void) *_bexpand(_segment *seg*, **void _based(void)** **memblock*, **size_t** *size*);

Include	MALLOC.H
seg	Based-heap segment selector
memblock	Pointer to previously allocated memory
size	New size in bytes
Returns	A based pointer to the resized memory block if successful; **_NULLOFF** if not

Changes the size of a block in the based heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bfree

void _bfree(_segment *seg*, **void _based(void)** **memblock*);

Include	MALLOC.H
seg	Segment selected
memblock	Allocated memory block
Returns	No return value

Frees a memory block in the based heap (previously allocated by a call to **_bmalloc** or **_brealloc**).

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bfreeseg

int _bfreeseg(_segment *seg* **);**

Include	MALLOC.H
seg	Based-heap segment selector
Returns	0 if successful; –1 in the case of an error

Frees a specified based-heap segment.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bheapadd

int _bheapadd(_segment *seg*, **void _based(void) ****memblock*, **size_t** *size* **);**

Include	MALLOC.H
seg	Based-heap segment selector
memblock	Pointer to based memory
size	Size in bytes of memory to add
Returns	A segment selector if successful; –1 if not

Adds an unused piece of memory to the specified based heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bheapchk

int _bheapchk(_segment *seg* **);**

Include	MALLOC.H
seg	Based-heap segment to check or **_NULLSEG** to check all based-heap segments
Returns	**_HEAPOK**, **_HEAPEMPTY**, **_HEAPBADBEGIN**, **_HEAPBADNODE**

Checks for minimal consistency in the based heap specified by the selector value *seg*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bheapmin

int _bheapmin(_segment *seg* **);**

Include	MALLOC.H
seg	Based-heap segment to minimize, or **_NULLSEG** for all based-heap segments
Returns	0 if successful; –1 if not

Releases unused memory in the based heap, minimizing the heap.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_bheapseg

_segment _bheapseg(size_t *size* **);**

Include	MALLOC.H
size	Segment size to allocate
Returns	The selector for the newly allocated segment if successful; –1 if not

Allocates a based-heap segment.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_bheapset

int _bheapset(_segment *seg*, **unsigned int** *fill* **);**

Include	MALLOC.H
seg	Based-heap segment selector or **_NULLSEG** for all based-heap segments
fill	Fill character
Returns	**_HEAPOK**, **_HEAPEMPTY**, **_HEAPBADBEGIN**, **_HEAPBADNODE**

Checks for minimal consistency in the based-heap segment specified by the selector value *seg*, and then sets the heap's free entries with the *fill* value.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_bheapwalk

int _bheapwalk(_segment *seg*, **_HEAPINFO** ***entryinfo*);**

Include	MALLOC.H
seg	Based-heap selector or **_NULLSEG** for all based-heap segments
entryinfo	Structure to contain information about the next heap entry, defined in MALLOC.H
Returns	**_HEAPOK**, **_HEAPEMPTY**, **_HEAPBADPTR**, **_HEAPBADBEGIN**, **_HEAPBADNODE**, **_HEAPEND**

Walks through the based-heap segment specified by the selector *seg* (or all based-heap segments if *seg* is equal to **_NULLSEG**), one entry per call, returning a pointer to a **_HEAPINFO** structure that contains information about the next based-heap entry.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bios_disk

unsigned _bios_disk(unsigned *service*, **struct diskinfo_t** ***diskinfo*);**

Include	BIOS.H
service	**_DISK_RESET**, **_DISK_STATUS**, **_DISK_READ**, **_DISK_WRITE**, **_DISK_VERIFY**, **_DISK_FORMAT**
diskinfo	Disk parameters structure defined in BIOS.H
Returns	The value stored in the AX register upon return from the BIOS interrupt

Calls BIOS disk-access services (INT 0x13).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_equiplist

unsigned _bios_equiplist(void);

Include BIOS.H

Returns A set of bits indicating what is installed

Calls INT 0x11 to determine what hardware and peripherals are currently installed on the machine.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_keybrd

unsigned _bios_keybrd(unsigned *service* **);**

Include BIOS.H

service **_KEYBRD_READ, _N_KEYBRD_READ, _KEYBRD_READY, _N_KEYBRD_READY, _KEYBRD_SHIFTSTATUS, _N_KEYBRD_SHIFTSTATUS**

Returns The value returned in the AX register upon return from the BIOS interrupt

Calls BIOS keyboard services (INT 0x16).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_memsize

unsigned _bios_memsize(void);

Include BIOS.H

Returns The total amount of installed memory in 1K blocks

Calls BIOS memory-size service (INT 0x12) to determine the total amount of main memory installed.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_printer

unsigned _bios_printer(unsigned *service*, **unsigned** *printer*, **unsigned** *data* **);**

Include	BIOS.H
service	**_PRINTER_WRITE, _PRINTER_INIT, _PRINTER_STATUS**
printer	Target printer port
data	Output data
Returns	The value returned in the AX register upon return from the BIOS interrupt

Calls BIOS printer services (INT 0x17) to perform printer tasks for parallel printers.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_serialcom

unsigned _bios_serialcom(unsigned *service*, **unsigned** *serial_port*, **unsigned** *data* **);**

Include	BIOS.H
service	**_COM_INIT, _COM_RECEIVE, _COM_SEND, _COM_STATUS**
serial_port	Serial port to use
data	**_COM_CHR7, _COM_CHR8, _COM_STOP1, _COM_STOP2, _COM_NOPARITY, _COM_EVENPARITY, _COM_ODDPARITY, _COM_110, _COM_150, _COM_300, _COM_600, _COM_1200, _COM_2400, _COM_4800, _COM_9600**
Returns	A 16-bit integer whose meaning varies depending on the service requested

Calls BIOS communications services (INT 0x14) to provide serial communications.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_timeofday

unsigned _bios_timeofday(unsigned *service*, **long** ***timeval**);

Include	BIOS.H
service	**_TIME_GETCLOCK, _TIME_SETCLOCK**
timeval	Clock count
Returns	The value returned in the AX register upon return from the BIOS interrupt

Calls BIOS time and date services (INT 0x1A) to get or set the current system clock count.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_bmalloc

void _based(void) *_bmalloc(_segment *seg*, **size_t** *size*);

Include	MALLOC.H
seg	Based-heap segment selector
size	Number of bytes to allocate
Returns	A **void** pointer to the allocated space; **_NULLOFF** if it fails

Allocates a memory block of at least *size* bytes in the based-heap segment specified by the selector *seg*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_bmsize

size_t _bmsize(_segment *seg*, **void _based(void) ***memblock* **);**

Include	MALLOC.H
seg	Based-heap segment selector
memblock	Pointer to based-heap memory block
Returns	The size in bytes

An unsigned integer representing the size in bytes of the based-heap memory block allocated by a call to **_bcalloc**, **_bmalloc**, or **_brealloc**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_brealloc

void _based(void) *_brealloc(_segment *seg*, **void _based(void) ***memblock*, **size_t** *size* **);**

Include	MALLOC.H
seg	Based-heap segment selector
memblock	Pointer to previously allocated memory block
size	New size in bytes
Returns	A **void _based** pointer to the reallocated memory block if successful; **_NULLOFF** if not

Changes the size of a previously allocated based-heap memory block, possibly relocating it.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

bsearch

void *bsearch(const void *_key_, const void *_base_, size_t _num_, size_t _width_,
int (*_compare_)(const void *_elem1_, const void *_elem2_));

Include	STDLIB.H, SEARCH.H
key	Pointer to object to search for
base	Pointer to base of search data
num	Number of elements
width	Width of elements
compare	Comparison function
elem1	Pointer to the key for the search
elem2	Pointer to the array element to be compared with the key
Returns	A pointer to the first occurrence of the object pointed to by *key* in the array that *base* points to; **NULL** if *key* is not found

Performs a binary search of a sorted array. Note that *key* is a pointer to an object of size *width*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

cabs

double cabs(struct complex *z* **);**

Include	MATH.H, ERRNO.H
z	Complex number structure defined in MATH.H
Returns	**HUGE_VAL** on overflow
errno	**ERANGE**

Calculates the absolute value of a complex number.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

cabsl

long double cabsl(struct _complexl *z* **);**

Include	MATH.H, ERRNO.H
z	Long double-precision complex number structure defined in MATH.H
Returns	The absolute value if successful; **_LHUGE_VAL** on overflow
errno	**ERANGE**

Calculates the absolute value of a complex number. Long double-precision floating-point version of **cabs**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

calloc

void *calloc(size_t *num*, **size_t** *size* **);**

Include	STDLIB.H or MALLOC.H
num	Number of elements
size	Length in bytes of each element
Returns	A pointer to the allocated space; **NULL** if there is insufficient memory available

Allocates storage for an array of *num* elements of *size* bytes. All bytes are initialized to 0.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ceil

double ceil(double *x* **);**

Include	MATH.H
x	Floating-point value
Returns	A value representing the smallest integer that is greater than or equal to the argument

Calculates the ceiling of a value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ceill

long double ceill(long double *x* **);**

Include	MATH.H
x	Long double-precision floating-point value
Returns	The long double result, rounded up

Returns a long double-precision floating-point value representing the smallest integer that is greater than or equal to the argument.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_cexit

void _cexit(void);

Include	PROCESS.H
Returns	No return value

Performs the **exit** termination procedures (e.g., flushing buffers) but returns control to the caller; does not terminate the process.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_c_exit

void _c_exit(void);

Include PROCESS.H

Returns No return value

Performs _exit termination procedures (i.e., without cleanup operations), but returns control to the caller; does not terminate the process.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

cgets

char *cgets(char *_buffer_);

Include CONIO.H

buffer Storage location for data

Returns A pointer to the start of a string of characters

Reads a string of characters (excluding terminal \n) directly from the console and stores the string and its length in *buffer*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_chain_intr

void _chain_intr(void(_interrupt _far *_target_)( ));

Include DOS.H

target Target interrupt routine

Returns No return value

Chains one interrupt handler to another interrupt handler.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

chdir

int chdir(char **dirname* **);**

Include	DIRECT.H, ERRNO.H
dirname	Path name of new working directory
Returns	0 if the working directory is successfully changed; –1 if the specified path name could not be found
errno	**ENOENT**

Changes the current working directory.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_chdrive

int _chdrive(int *drive* **);**

Include	DIRECT.H
drive	Number of new working drive (A: = 1, B: = 2)
Returns	0 if the working drive is successfully changed; –1 if not

Changes the current working drive.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

chmod

int chmod(char **filename*, **int** *pmode* **);**

Include	SYS\TYPES.H, SYS\STAT.H, IO.H, ERRNO.H
filename	Path name of existing file
pmode	Permission setting for file
Returns	0 if the permission setting is successfully changed; **ENOENT** if the specified file could not be found
errno	**ENOENT**

Changes file permission settings.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

chsize

int chsize(int *handle*, **long** *size* **);**

Include	IO.H, ERRNO.H
handle	Handle referring to open file
size	New length of file in bytes
Returns	0 if the file size is successfully changed; –1 if not
errno	**EACCESS, EBADF, ENOSPC**

Changes the size of a file.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_clear87

unsigned int _clear87(void);

Include	FLOAT.H
Returns	A word whose bits indicate the floating-point status

Gets and clears a floating-point status word.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

clearerr

void clearerr(FILE ***stream*);**

Include	STDIO.H
stream	Pointer to **FILE** structure
Returns	No return value

Resets the error and end-of-file indicators for a stream.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_clearscreen

void _far _clearscreen(short *area* **);**

Include	GRAPH.H
area	**_GCLEARSCREEN**, **_GVIEWPORT**, **_GWINDOW**
Returns	No return value

Clears the screen and fills it with the current background color.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

clock

clock_t clock(void);

Include	TIME.H
Returns	The elapsed processor time; –1, cast as **clock_t**, if the processor time is not available

Calculates how much processor time in seconds has been used by the calling process.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

close

int close(int *handle* **);**

Include	IO.H, ERRNO.H
handle	Handle referring to open file
Returns	0 if the file was successfully closed; –1 if the file-handle argument is invalid
errno	**EBADF**

Closes a file.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_control87

unsigned int _control87(unsigned int *new*, **unsigned int** *mask* **);**

Include	FLOAT.H
new	New control-word bit values
mask	Mask for new control-word bits to set
Returns	A word whose bits indicate the floating-point control state

Gets and sets the floating-point control word.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

cos

double cos(double *x* **);**

Include	MATH.H, ERRNO.H
x	Radians
Returns	The cosine of *x*; 0 if *x* is so large that significance is completely lost
errno	**ERANGE**

Calculates the cosine of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

cosh

double cosh(double *x* **);**

Include	MATH.H, ERRNO.H
x	Radians
Returns	The hyperbolic cosine of *x*; 0 if *x* is so large that significance is completely lost; **HUGE_VAL** if the result is too large
errno	**ERANGE**

Calculates the hyperbolic cosine of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

coshl

long double coshl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Radians
Returns	The hyperbolic cosine of *x* if successful; 0 if *x* is so large that significance is completely lost; **_LHUGE_VAL** if the result is too large
errno	**ERANGE**

Calculates the hyperbolic cosine of *x*. Long double-precision floating-point version of **cosh**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

cosl

long double cosl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Radians
Returns	The cosine of *x* if successful; 0 if *x* is so large that significance is completely lost
errno	**ERANGE**

Calculates the cosine of *x*. Long double-precision floating-point version of **cos**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

cprintf

int cprintf(char **format* [[, *argument*]]... **);**

Include	CONIO.H
format	Format-control string
argument	Optional arguments
Returns	The number of characters printed

Formats and prints to the console. See **printf/scanf** format table.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

cputs

int cputs(char **string* **);**

Include	CONIO.H
string	Output string
Returns	0 if successful; a nonzero value if not

Writes a null-terminated string directly to the console.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

creat

int creat(char **filename*, **int** *pmode* **);**

Include	SYS\TYPES.H, SYS\STAT.H, IO.H, ERRNO.H
filename	Path name of new file
pmode	**S_IWRITE, S_IREAD, S_IREAD|S_IWRITE**
Returns	A handle for the created file if successful; –1 if not
errno	**EACCESS, EMFILE, ENOENT**

Creates a new file or opens and truncates an existing file.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

cscanf

int cscanf(char **format* [[, *argument*]]... **);**

Include	CONIO.H
format	Format-control string
argument	Optional arguments
Returns	The number of fields that were successfully converted and assigned; **EOF** for an attempt to read at end-of-file

Reads formatted data directly from the console into a specified location. See **printf/scanf** format table.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

ctime

char *ctime(const time_t *_timer_);

Include	TIME.H
timer	Pointer to stored time
Returns	A pointer to the character string result; **NULL** if *timer* represents a date before 1980

Converts time stored as a **time_t** value to a character string.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

cwait

int cwait(int *_termstat_, int _procid_, int _action_);

Include	PROCESS.H, ERRNO.H
termstat	Termination status and return code
procid	Child process identification
action	**WAIT_CHILD, WAIT_GRANDCHILD**
Returns	The child process identification after normal termination; −1 otherwise
errno	**EINTR, EINVAL, ECHILD**

Suspends the calling process until the child process terminates, and updates the *termstat* variable.

□ ANSI □ DOS ■ OS/2 □ UNIX □ XENIX

dieeetomsbin

int dieeetomsbin(double *_src8_, double *_dst8_);

Include	MATH.H
src8	Buffer containing number to convert
dst8	Buffer for converted number
Returns	0 if the conversion is successful; 1 if the conversion causes an overflow

Converts a double-precision number in IEEE format to Microsoft binary format.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

difftime

double difftime(time_t *timer1*, **time_t** *timer0* **);**

Include	TIME.H
timer1, *timer0*	Time value structures defined in TIME.H
Returns	The difference between two times

Computes the elapsed time in seconds between two time values.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_disable

void _disable(void);

Include	DOS.H
Returns	No return value

Disables interrupts by executing an 8086 CLI machine instruction.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_displaycursor

short _far _displaycursor(short *toggle* **);**

Include	GRAPH.H
toggle	**_GCURSOROFF**, **_GCURSORON**
Returns	The previous value of *toggle*

Determines whether the cursor will be displayed.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

div

div_t div(int *numer*, **int** *denom* **);**

Include	STDLIB.H
numer	Numerator
denom	Denominator
Returns	The quotient and remainder in a structure defined in STDLIB.H

Divides the numerator by the denominator computing the quotient and remainder.

■ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

dmsbintoieee

int dmsbintoieee(double ***src8*, double** ****dst8* **);**

Include	MATH.H
src8	Buffer containing number to convert
dst8	Buffer for converted number
Returns	0 if the conversion is successful; 1 if the conversion causes an overflow

Converts a double-precision number in Microsoft binary format to IEEE format.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_dos_allocmem

unsigned _dos_allocmem(unsigned *size*, **unsigned ****seg* **);**

Include	DOS.H, ERRNO.H
size	Block size to allocate
seg	Segment descriptor return buffer
Returns	0 if successful; the DOS error code if not
errno	**ENOMEM**

Allocates a block of memory *size* paragraphs long (16 bytes each).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_close

unsigned _dos_close(int *handle* **);**

Include	DOS.H, ERRNO.H
handle	Target file handle
Returns	0 if successful; the DOS error code if not
errno	**EBADF**

Uses system call 0x3E to close a file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_creat

unsigned _dos_creat(char ***filename*****, unsigned** *attrib*, **int** ***handle**);

Include	DOS.H, ERRNO.H
filename	File path name
attrib	**_A_ARCH, _A_HIDDEN, _A_NORMAL, _A_RDONLY, _A_SUBDIR, _A_SYSTEM, _A_VOLID**
handle	Handle return buffer
Returns	0 if successful; the DOS error code if not
errno	**ENOENT, EMFILE, EACCESS, EEXIST**

Uses system call 0x3C to create a new file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_creatnew

unsigned _dos_creatnew(char ***filename*****, unsigned** *attrib*, **int** ***handle**);

Include	DOS.H, ERRNO.H
filename	File path name
attrib	**_A_ARCH, _A_HIDDEN, _A_NORMAL, _A_RDONLY, _A_SUBDIR, _A_SYSTEM, _A_VOLID**
handle	Handle return buffer
Returns	0 if successful; the DOS error code if not
errno	**ENOENT, EMFILE, EACCESS, EEXIST**

Uses system call 0x5B to create a new file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

dosexterr

int dosexterr(struct DOSERROR **errorinfo* **);**

Include	DOS.H
errorinfo	Error-information structure defined in DOS.H
Returns	The value in the AX register (identical to the value in the **exterror** structure field)

Uses system call 0x59 to get extended error information.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_findfirst

unsigned _dos_findfirst(char **filename*, **unsigned** *attrib*, **struct find_t** **fileinfo* **);**

Include	DOS.H, ERRNO.H
filename	Target file name
attrib	**_A_ARCH, _A_HIDDEN, _A_NORMAL, _A_RDONLY, _A_SUBDIR, _A_SYSTEM, _A_VOLID**
fileinfo	File-information return structure defined in DOS.H
Returns	0 if successful; the DOS error code if not
errno	**ENOENT**

Uses system call 0x4E to return information about the first instance of a file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_findnext

unsigned _dos_findnext(struct find_t *_fileinfo_);

Include	DOS.H, ERRNO.H
fileinfo	File-information return structure defined in DOS.H
Returns	0 if successful; the DOS error code if not
errno	**ENOENT**

Uses system call 0x4F to find the next file that matches the name and attributes specified in a prior call to **_dos_findfirst**.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_freemem

unsigned _dos_freemem(unsigned *seg* **);**

Include	DOS.H, ERRNO.H
seg	Block to be released
Returns	0 if successful; the DOS error code if not
errno	**ENOMEM**

Uses system call 0x49 to release a block of memory previously allocated by **_dos_allocmem**.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getdate

void _dos_getdate(struct dosdate_t *_date_);

Include	DOS.H
date	Current date structure defined in DOS.H
Returns	No return value

Uses system call 0x2A to get the current system date.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getdiskfree

unsigned _dos_getdiskfree(unsigned *drive*, **struct diskfree_t** **diskspace* **);**

Include	DOS.H, ERRNO.H
drive	Target disk drive
diskspace	Disk-information structure defined in DOS.H
Returns	0 if successful; a nonzero value if not
errno	**EINVAL**

Uses system call 0x36 to get information on a disk drive.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getdrive

void _dos_getdrive(unsigned **drive* **);**

Include	DOS.H
drive	Current drive return buffer
Returns	No return value

Uses system call 0x19 to get the current disk drive (A: = 0, B: = 1).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getfileattr

unsigned _dos_getfileattr(char **pathname*, **unsigned** **attrib* **);**

Include	DOS.H, ERRNO.H
pathname	Full path of target file or directory
attrib	**_A_ARCH, _A_HIDDEN, _A_NORMAL, _A_RDONLY, _A_SUBDIR, _A_SYSTEM, _A_VOLID**
Returns	0 if successful; the DOS error code if not
errno	**ENOENT**

Uses system call 0x43 to get the current attributes of a file or directory.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getftime

unsigned _dos_getftime(int *handle*, **unsigned** **date*, **unsigned** **time* **);**

Include	DOS.H, ERRNO.H
handle	Target file
date	Date-return buffer
time	Time-return buffer
Returns	0 if successful; the DOS error code if not
errno	**EBADF**

Uses system call 0x57 to get the time and date a file was last written.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_gettime

void _dos_gettime(struct dostime_t *_time_);

Include	DOS.H
time	Current system time structure defined in DOS.H
Returns	No return value

Uses system call 0x2C to get the current system time.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getvect

void(_interrupt _far *_dos_getvect(unsigned *intnum* **))( );**

Include	DOS.H
intnum	Target interrupt vector
Returns	Far pointer to the current interrupt handler

Uses system call 0x35 to get the current value of an interrupt vector.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_keep

void _dos_keep(unsigned *retcode*, **unsigned** *memsize* **);**

Include	DOS.H
retcode	Exit status code
memsize	Allocated resident memory in 16-byte paragraphs
Returns	No return value

Uses system call 0x31 to install a terminate-and-stay-resident (TSR) program in memory.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_open

unsigned _dos_open(char *_filename_, unsigned _mode_, int *_handle_);

Include	DOS.H, FCNTL.H, SHARE.H, ERRNO.H
filename	Path name to an existing file
mode	**O_RDONLY, O_WRONLY, O_RDWR, SH_COMPAT, SH_DENYRW, SH_DENYWR, SH_DENYRD, SH_DENYNO, O_NOINHERIT**
handle	Handle-return buffer
Returns	0 if successful; the DOS error code if not
errno	**EINVAL, ENOENT, EMFILE, EACCESS**

Uses system call 0x3D to open an existing file, with modes specified by *mode*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_read

unsigned _dos_read(int _handle_, void _far *_buffer_, unsigned _count_, unsigned *_numread_);

Include	DOS.H, ERRNO.H
handle	File to read
buffer	Buffer to write to
count	Number of bytes to read
numread	Number of bytes actually read
Returns	0 if successful; the DOS error code if not
errno	**EACCESS, EBADF**

Uses system call 0x3F to read data from a file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setblock

unsigned _dos_setblock(unsigned *size*, **unsigned** *seg*, **unsigned ****maxsize* **);**

Include	DOS.H, ERRNO.H
size	New segment size
seg	Target segment
maxsize	Maximum size buffer
Returns	0 if successful; the DOS error code if not
errno	**ENOMEM**

Changes the size of a segment previously allocated by **_dos_allocmem**.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setdate

unsigned _dos_setdate(struct dosdate_t ****date* **);

Include	DOS.H, ERRNO.H
date	System date structure defined in DOS.H
Returns	0 if successful; a nonzero value if not
errno	**EINVAL**

Uses system call 0x2B to set the current system date.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setdrive

void _dos_setdrive(unsigned *drive*, **unsigned** **numdrives* **);**

Include	DOS.H
drive	New default drive
numdrives	Total drives available
Returns	No return value

Uses system call 0x0E to set the current default drive.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setfileattr

unsigned _dos_setfileattr(char **pathname*, **unsigned** *attrib* **);**

Include	DOS.H, ERRNO.H
pathname	Full path of target file or directory
attrib	**_A_ARCH, _A_HIDDEN, _A_NORMAL, _A_RDONLY, _A_SUBDIR, _A_SYSTEM, _A_VOLID**
Returns	0 if successful; the DOS error code if not
errno	**EACCESS, ENOENT**

Uses system call 0x43 to set the attributes of a file or a directory.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setftime

unsigned _dos_setftime(int *handle*, **unsigned** *date*, **unsigned** *time* **);**

Include	DOS.H, ERRNO.H
handle	Target file
date	Date of last write
time	Time of last write
Returns	0 if successful; the DOS error code if not
errno	**EBADF**

Uses system call 0x57 to set the date and time a file was last written.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_settime

unsigned _dos_settime(struct dostime_t * *time* **);**

Include	DOS.H, ERRNO.H
time	System time structure defined in DOS.H
Returns	0 if successful; a nonzero value if not
errno	**EINVAL**

Uses system call 0x2D to set the current system time.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setvect

void _dos_setvect(unsigned *intnum*, **void(_interrupt _far *** *handler*)( ));

Include	DOS.H
intnum	Target interrupt vector
handler	Interrupt handler to assign
Returns	No return value

Uses system call 0x25 to set the current value of an interrupt vector.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_write

unsigned _dos_write(int *handle*, **void _far *** *buffer*, **unsigned** *count*, **unsigned *** *numwrt*);

Include	DOS.H, ERRNO.H
handle	File to write to
buffer	Buffer to write from
count	Number of bytes to write
numwrt	Number of bytes actually written
Returns	0 if successful; the DOS error code if not
errno	**EACCESS, EBADF**

Uses system call 0x40 to write data to a file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

dup

int dup(int *handle* **);**

Include	IO.H, ERRNO.H
handle	Handle of an open file
Returns	A new file handle if successful; –1 if not
errno	**EBADF, EMFILE**

Associates a second file handle with the currently open file.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

dup2

int dup2(int *handle1*, **int** *handle2* **);**

Include	IO.H, ERRNO.H
handle1	Handle of an open file
handle2	Any handle value
Returns	0 if successful; –1 if not
errno	**EBADF, EMFILE**

Associates the specified second handle with the currently open file.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ecvt

char *ecvt(double *value*, **int** *count*, **int** **dec*, **int** **sign*);

Include	STDLIB.H
value	Number to be converted
count	Number of digits stored
dec	Stored decimal-point position
sign	Sign of converted number
Returns	A pointer to the string of digits

Converts a floating-point number to a character string.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_ellipse

short _far _ellipse(short *control*, **short** *x1*, **short** *y1*, **short** *x2*, **short** *y2*);

Include	GRAPH.H
control	**_GFILLINTERIOR**, **_GBORDER**
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws an ellipse using view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_ellipse_w

short _far _ellipse_w(short *control*, **double** *wx1*, **double** *wy1*, **double** *wx2*, **double** *wy2*);

Include	GRAPH.H
control	**_GFILLINTERIOR**, **_GBORDER**
wx1, *wy1*	Upper left corner of bounding rectangle
wx2, *wy2*	Lower right corner of bounding rectangle
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws an ellipse using window coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_ellipse_wxy

short _far _ellipse_wxy(short *control*, **struct _wxycoord _far** **pwxy1*,
struct _wxycoord _far **pwxy2*);

Include	GRAPH.H
control	**_GFILLINTERIOR**, **_GBORDER**
pwxy1	Upper left corner of bounding rectangle
pwxy2	Lower right corner of bounding rectangle
Returns	A nonzero value if successful; 0 if not

Draws an ellipse using window coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_enable

void _enable(void);

Include DOS.H

Returns No return value

Enables interrupts by executing an 8086 STI machine instruction.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_endthread

void _far _endthread(void);

Include PROCESS.H

Returns No return value

Terminates a thread created by **_beginthread**.

☐ ANSI ☐ DOS ■ OS/2 ☐ UNIX ☐ XENIX

eof

int eof(int *handle* **);**

Include IO.H, ERRNO.H

handle Handle referring to open file

Returns 1 if the current position is end-of-file; 0 if not; –1 if an error has occurred

errno **EBADF**

Determines whether the end of the file has been reached.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

execl

int execl(char **cmdname*, **char** **arg0*, ... **char** **argn*, **NULL**);

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
arg0, ... argn	List of pointers to arguments, terminated by **NULL**
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed individually.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execle

int execle(char **cmdname*, **char** **arg0*, ... **char** **argn*, **NULL, char *****envp*);

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
arg0, ... argn	List of pointers to arguments, terminated by **NULL**
envp	Array of pointers to environment settings
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed individually. An array of pointers to environment arguments is explicitly passed to the child process.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execlp

int execlp(char **cmdname*, **char** **arg0*, ... **char** **argn*, **NULL** **);**

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
arg0, ... argn	List of pointers to arguments, terminated by **NULL**
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed individually. Uses the PATH environment variable to find the file to be executed.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execlpe

int execlpe(char **cmdname*, **char** **arg0*, ... **char** **argn*, **NULL, char** ***envp* **);**

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
arg0, ... argn	List of pointers to arguments, terminated by **NULL**
envp	Array of pointers to environment settings
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed individually. Uses the PATH environment variable to find the file to be executed. An array of pointers to environment arguments is explicitly passed to the child process.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execv

int execv(char *cmdname*, **char *****argv* **);**

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed as an array of pointers.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execve

int execve(char *cmdname*, **char *****argv*, **char *****envp* **);**

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
envp	Array of pointers to environment settings
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed as an array of pointers. An array of pointers to environment arguments is explicitly passed to the child process.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execvp

int execvp(char *cmdname*, **char ****argv*);**

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed as an array of pointers. Uses the PATH environment variable to find the file to be executed.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execvpe

int execvpe(char *cmdname*, **char ****argv*, **char ****envp*);**

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
envp	Array of pointers to environment settings
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed as an array of pointers. Uses the PATH environment variable to find the file to be executed. An array of pointers to environment arguments is explicitly passed to the child process.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

exit

void exit(int *status* **);**

Include	PROCESS.H or STDLIB.H
status	Exit status
Returns	No return value

Terminates a process after normal cleanup operations (e.g., flushing buffers).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_exit

void _exit(int *status* **);**

Include	PROCESS.H or STDLIB.H
status	Exit status
Returns	No return value

Terminates a process without normal cleanup operations.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

exp

double exp(double *x* **);**

Include	MATH.H, ERRNO.H
x	Floating-point value
Returns	Exponential function of *x* if successful; 0 on underflow; **HUGE_VAL** on overflow
errno	**ERANGE**

Returns the exponential function of the floating-point argument *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_expand

void *_expand(void *_memblock_, size_t _size_);

Include	MALLOC.H
memblock	Pointer to previously allocated memory block
size	New size in bytes
Returns	A pointer to the resized memory block; **NULL** if there is insufficient memory to expand the block

Changes the size of a previously allocated memory block by attempting to expand or contract the block without moving its location in the heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

expl

long double expl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
Returns	Exponential function of *x* if successful; 0 on underflow; **_LHUGE_VAL** on overflow
errno	**ERANGE**

Returns the exponential function of the floating-point argument *x*. Long double-precision floating-point version of **exp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fabs

double fabs(double *x* **);**

Include	MATH.H
x	Floating-point value
Returns	The absolute value of *x*

Gets the absolute value of a double-precision floating-point value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fabsl

long double fabsl(long double *x* **);**

Include	MATH.H
x	Long double-precision floating-point value
Returns	The absolute value of *x*

Gets the absolute value of a long double-precision floating-point value.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fcalloc

void _far *_fcalloc(size_t *num*, **size_t** *size* **);**

Include	MALLOC.H
num	Number of elements
size	Length in bytes of each element
Returns	A far pointer to the allocated space if successful; **NULL** if there is insufficient memory available

Allocates storage space for an array in the far heap.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fclose

int fclose(FILE *_stream_);

Include	STDIO.H
stream	Target stream
Returns	0 if the stream is successfully closed; **EOF** if not

Closes an open stream.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fcloseall

int fcloseall(void);

Include	STDIO.H
Returns	The total number of streams closed; **EOF** if an error occurs

Closes all open streams.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fcvt

char *fcvt(double *value*, **int** *count*, **int *** *dec*, **int *** *sign* **);**

Include	STDLIB.H
value	Number to be converted
count	Number of digits after decimal point
dec	Pointer to stored decimal-point position
sign	Pointer to stored sign indicator
Returns	A pointer to the string of digits

Converts a floating-point number to a character string.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fdopen

FILE *fdopen(int *handle*, **char** **mode* **);**

Include	STDIO.H
handle	Handle of open file
mode	Access permissions: **"r"**, **"w"**, **"a"**, **"r+"**, **"w+"**, **"a+"**, **"t"**, **"b"**
Returns	A pointer to the open stream if successful; **NULL** if not

Associates a stream with a file handle, allowing a file opened for low-level I/O to be buffered and formatted.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

feof

int feof(FILE **stream* **);**

Include	STDIO.H
stream	Pointer to a stream
Returns	A nonzero value after the first read operation that attempts to read past the end of the file; 0 if the current position is not end-of-file

Determines whether the end of *stream* has been reached.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ferror

int ferror(FILE **stream* **);**

Include	STDIO.H
stream	Pointer to a stream
Returns	A nonzero value if an error occurred; 0 if not

Tests for a reading or writing error on *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_fexpand

void _far *_fexpand(void _far *_memblock_, size_t _size_);

Include	MALLOC.H
memblock	Pointer to previously allocated memory
size	New size in bytes
Returns	A far pointer to the resized memory block if successful; **NULL** if not

Changes the size of a block in the far heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fflush

int fflush(FILE *_stream_);

Include	STDIO.H
stream	Pointer to target stream
Returns	0 if successful or where the specified stream has no buffer or is open for reading only; **EOF** otherwise

Writes the contents of the buffer associated with the stream to the associated file.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_ffree

void _ffree(void _far *_memblock_);

Include	MALLOC.H
memblock	Allocated memory block in the far heap
Returns	No return value

Frees a memory block in the far heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fgetc

int fgetc(FILE *_stream_);

Include	STDIO.H
stream	Pointer to target stream
Returns	The character read; **EOF** if an error occurred or at end-of-file

Reads a character from *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fgetchar

int fgetchar(void);

Include	STDIO.H
Returns	The character read; **EOF** if an error occurred or at end-of-file

Reads a character from the standard input stream, **stdin**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fgetpos

int fgetpos(FILE *_stream_, fpos_t *_pos_);

Include	STDIO.H, ERRNO.H
stream	Pointer to target stream
pos	Position-indicator buffer
Returns	0 if successful; a nonzero value if not
errno	**EBADF, EINVAL**

Gets the current value of the *stream* file-position indicator.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fgets

char *fgets(char *_string_, int _n_, FILE *_stream_);

Include	STDIO.H
string	Storage location for data
n	Maximum number of characters read and stored
stream	Pointer to target stream
Returns	*string* if successful; **NULL** to indicate an error or end-of-file

Reads a string of, at most, *n* characters from the input *stream* and stores it in *string*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_fheapchk

int _fheapchk(void);

Include	MALLOC.H
Returns	**_HEAPOK**, **_HEAPEMPTY**, **_HEAPBADBEGIN**, **_HEAPBADNODE**

Checks for minimal consistency in the far heap.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fheapmin

int _fheapmin(void);

Include	MALLOC.H
Returns	0 if successful; –1 if not

Releases unused memory in the far heap back to the operating system.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fheapset

int _fheapset(unsigned int *fill* **);**

Include	MALLOC.H
fill	Fill character
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADBEGIN, _HEAPBADNODE**

Checks for minimal consistency in the far heap and then sets the heap's free entries with the *fill* value.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fheapwalk

int _fheapwalk(_HEAPINFO *****entryinfo* **);

Include	MALLOC.H
entryinfo	Structure to contain information about the next far heap entry, defined in MALLOC.H
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADPTR, _HEAPBADBEGIN, _HEAPBADNODE, _HEAPEND**

Walks through the far heap, one entry per call, returning a pointer to a **_HEAPINFO** structure that contains information about the next far heap entry.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fieeetomsbin

int fieeetomsbin(float **src4*, **float** **dst4* **);**

Include	MATH.H
src4	Value to convert
dst4	Buffer for converted value
Returns	0 if successful; 1 if the conversion causes an overflow

Converts a single-precision floating-point number in IEEE format to Microsoft binary format.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

filelength

long filelength(int *handle* **);**

Include	IO.H, ERRNO.H
handle	Target file handle
Returns	The file length in bytes if successful; –1 if not
errno	**EBADF**

Gets the length in bytes of a file.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fileno

int fileno(FILE **stream* **);**

Include	STDIO.H
stream	Pointer to target stream
Returns	The file handle

Returns the file handle currently associated with *stream*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_floodfill

short _far _floodfill(short *x*, **short** *y*, **short** *boundary*);

Include	GRAPH.H
x, *y*	Starting point
boundary	Fill boundary color
Returns	A nonzero value if successful; 0 if the fill could not be completed, the starting point lies on the *boundary* color, or the starting point lies outside the clipping region

Fills an area of the display beginning at the view coordinate point using the current color and fill mask.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_floodfill_w

short _far _floodfill_w(double *wx*, **double** *wy*, **short** *boundary*);

Include	GRAPH.H
wx, *wy*	Starting point
boundary	Fill boundary color
Returns	A nonzero value if successful; 0 if the fill could not be completed, the starting point lies on the *boundary* color, or the starting point lies outside the clipping region

Fills an area of the window using the current color and fill mask.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

floor

double floor(double *x* **);**

Include	MATH.H
x	Floating-point value
Returns	The floating-point result

Returns a floating-point value representing the largest integer that is less than or equal to *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

floorl

long double floorl(long double *x* **);**

Include	MATH.H
x	Long double-precision floating-point value
Returns	Long double-precision floating-point result, rounded down

Returns a long double-precision floating-point value representing the largest integer that is less than or equal to *x*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

flushall

int flushall(void);

Include	STDIO.H
Returns	The number of open streams

Writes the contents of all buffers for open output streams to their files and clears all buffers for open input streams of their contents.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fmalloc

void _far *_fmalloc(size_t *size* **);**

Include	MALLOC.H
size	Number of bytes to allocate
Returns	A far pointer to the allocated space; **NULL** if it fails

Allocates a memory block of at least *size* bytes in the far heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemccpy

void _far * _far _fmemccpy(void _far ****dest*, **void _far ****src*, **int *c*, **unsigned int** *count* **);**

Include	STRING.H or MEMORY.H
dest	Far pointer to destination
src	Far pointer to source
c	Last character to copy
count	Number of characters
Returns	A far pointer to the byte in *dest* that immediately follows the copied character (*c*) if successful; **NULL** if *count* bytes copied before *c* was found

Copies zero or more far bytes of *src* to *dest*, copying up to and including the first occurrence of the character *c* or until *count* bytes have been copied, whichever comes first. Model-independent version of **memccpy**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemchr

void _far * _far _fmemchr(const void _far *_buf_, int _c_, size_t _count_);

Include	STRING.H or MEMORY.H
buf	Far pointer to buffer
c	Character to copy
count	Number of characters
Returns	A far pointer to the first location of *c* in *buf* if successful; **NULL** if not

Looks for the first occurrence of *c* in the first *count* bytes of *buf*. Model-independent version of **_memchr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemcmp

int _far _fmemcmp(const void _far *_buf1_, const void _far *_buf2_, size_t _count_);

Include	STRING.H or MEMORY.H
buf1, *buf2*	Far pointers to buffers to compare
count	Number of characters
Returns	A value less than, equal to, or greater than 0, depending on whether the size of the buffer pointed to by *buf1* is less than, equal to, or greater than the size of the buffer pointed to by *buf2*

Compares the first *count* bytes of two buffers and returns a value indicating their relationship. Model-independent version of **memcmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemcpy

void _far * _far _fmemcpy(void _far *_dest_, const void _far *_src_, size_t _count_);

Include	STRING.H or MEMORY.H
dest	Far pointer to the destination buffer
src	Far pointer to the buffer to copy from
count	Number of characters to copy
Returns	*dest*

Copies *count* bytes of *src* to *dest*. Model-independent version of **memcpy**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemicmp

int _far _fmemicmp(void _far *_buf1_, void _far *_buf2_, unsigned int _count_);

Include	STRING.H or MEMORY.H
buf1, *buf2*	Far pointers to buffers to compare
count	Number of characters
Returns	A value less than, equal to, or greater than 0, depending on whether the size of the buffer pointed to by *buf1* is less than, equal to, or greater than the size of the buffer pointed to by *buf2*

Compares a specified number of characters from two buffers without regard to letter case. Model-independent version of **memicmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemmove

void _far * _far _fmemmove(void _far *_dest_, const void _far *_src_, size_t _count_);

Include	STRING.H
dest	Far pointer to destination buffer
src	Far pointer to source buffer
count	Number of characters to copy
Returns	The value of *dest*

Copies *count* characters from *src* to *dest*. Model-independent version of **memmove**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemset

void _far * _far _fmemset(void _far *_dest_, int _c_, size_t _count_);

Include	STRING.H or MEMORY.H
dest	Far pointer to destination
c	Character to set
count	Number of characters
Returns	A far pointer to *dest*

Sets the first *count* bytes of *dest* to the character *c*. Model-independent version of **memset**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fmod

double fmod(double *x*, **double** *y* **);**

Include	MATH.H
x, *y*	Floating-point values
Returns	The floating-point remainder

Calculates the floating-point remainder *f* of (*x*, *y*) such that $x = i * y + f$, where *i* is an integer, *f* has the same sign as *x*, and the absolute value of *f* is less than the absolute value of *y*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fmodl

long double fmodl(long double *x*, **long double** *y* **);**

Include	MATH.H
x, *y*	Floating-point values
Returns	The floating-point remainder

Calculates the long double-precision floating-point remainder *f* of (*x*, *y*) such that $x = i * y + f$, where *i* is an integer, *f* has the same sign as *x*, and the absolute value of *f* is less than the absolute value of *y*. Long double-precision floating-point version of **fmod**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fmsbintoieee

int fmsbintoieee(float ****src4*, **float ****dst4* **);

Include	MATH.H
src4	Value to convert
dst4	Buffer for converted value
Returns	0 if successful; 1 if the conversion causes an overflow

Converts a floating-point number in Microsoft binary format to IEEE format.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fmsize

size_t _fmsize(void _far ***memblock*** **);**

Include	MALLOC.H
memblock	Pointer to far memory block
Returns	An unsigned integer representing the size in bytes of the far-heap memory block allocated by a call to **_fcalloc**, **_fmalloc**, or **_frealloc**

Returns the size of the memory block pointed to by *memblock*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fopen

FILE *fopen(const char ***filename*****, const char** ***mode*** **);**

Include	STDIO.H
filename	Path name of file
mode	Access permissions: **"r"**, **"r+"**, **"r+b"**, **"r+t"**, **"w"**, **"w+"**, **"w+b"**, **"w+t"**, **"a"**, **"a+"**, **"a+b"**, **"a+t"**
Returns	A pointer to the open file if successful; **NULL** if not

Opens a specified file for stream I/O.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

FP_OFF

unsigned FP_OFF(void _far ***address*** **);**

Include	DOS.H
address	Far pointer to memory address
Returns	An offset

Sets or gets the offset portion of a far address.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fpreset

void _fpreset(void);

Include	FLOAT.H
Returns	No return value

Reinitializes the floating-point math package.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fprintf

int fprintf(FILE *_stream_, const char *_format_ [, _argument_]...);

Include	STDIO.H
stream	Pointer to target stream
format	Format-control string
argument	Optional arguments
Returns	The number of characters printed if successful; a negative value if not

Formats and prints a series of characters and values to the output *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

FP_SEG

unsigned FP_SEG(void _far *_address_);

Include	DOS.H
address	Far pointer to memory address
Returns	A segment

Sets or gets the segment selector portion of a far address.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fputc

int fputc(int *c*, **FILE** ***stream*);**

Include	STDIO.H
c	Character to be written
stream	Pointer to target stream
Returns	The character written if successful; **EOF** if not

Writes a single character to the target stream.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fputchar

int fputchar(int *c* **);**

Include	STDIO.H
c	Character to be written
Returns	The character written if successful; **EOF** if not

Writes a single character to the standard output stream, **stdout**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fputs

int fputs(const char ****string*, **FILE** ****stream*);**

Include	STDIO.H
string	Output string
stream	Pointer to target stream
Returns	A nonnegative value if successful; **EOF** if not

Copies a string to the target stream.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fread

size_t fread(void ***buffer*****, size_t** *size*, **size_t** *count*, **FILE** ***stream*);**

Include	STDIO.H
buffer	Storage location for data
size	Item size in bytes
count	Maximum number of items to be read
stream	Pointer to target stream
Returns	The number of items actually read, which may be less than *count* if an error occurs

Reads up to *count* items of *size* bytes from the input *stream* and stores them in *buffer*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_frealloc

void _far *_frealloc(void _far ***memblock*****, size_t** *size* **);**

Include	MALLOC.H
memblock	Pointer to previously allocated far memory block
size	New size in bytes
Returns	A far pointer to the reallocated memory block if successful; **NULL** if not

Changes the size of a previously allocated far memory block, possibly moving it.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

free

void free(void **memblock* **);**

Include	STDLIB.H, MALLOC.H
memblock	Allocated memory block
Returns	No return value

Frees a memory block previously allocated by **calloc**, **malloc**, or **realloc**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_freect

unsigned int _freect(size_t *size* **);**

Include	MALLOC.H
size	Item size in bytes
Returns	The number of items that can be allocated in the near heap

Determines how much memory is available for dynamic memory allocation in the near heap by returning the number of times an item can be allocated.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

freopen

FILE *freopen(const char *_filename_, const char *_mode_, FILE *_stream_);

Include	STDIO.H
filename	Path name of new file
mode	Access permission: **"r"**, **"r+"**, **"r+b"**, **"r+t"**, **"w"**, **"w+"**, **"w+b"**, **"w+t"**, **"a"**, **"a+"**, **"a+b"**, **"a+t"**
stream	Pointer to target stream
Returns	A pointer to the newly opened file if successful; NULL if not

Closes the file currently associated with *stream* and reassigns *stream* to the file specified by *filename*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

frexp

double frexp(double *x*, **int** **expptr* **);**

Include	MATH.H
x	Floating-point value
expptr	Pointer to stored integer exponent
Returns	The mantissa if successful; 0 if *x* is 0

Breaks down the floating-point value (x) into a mantissa (m) and an exponent (n) such that the absolute value of m is greater than or equal to 0.5 and less than 1.0, and $x = m * 2^n$.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

frexpl

long double frexpl(long double *x*, **int** **expptr* **);**

Include	MATH.H
x	Long double-precision floating-point value
expptr	Pointer to stored integer exponent
Returns	The mantissa if successful; 0 if *x* is 0

Breaks down the long double-precision floating-point value (x) into a mantissa (m) and an exponent (n) such that the absolute value of m is greater than or equal to 0.5 and less than 1.0, and $x = m * 2^n$. Long double-precision floating-point version of **frexp**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fscanf

int fscanf(FILE **stream*, **const char** **format* [, *argument*]... **);**

Include	STDIO.H
stream	Pointer to target stream
format	Format-control string
argument	Optional arguments
Returns	The number of fields that were successfully converted and assigned, or **EOF** for end-of-file

Reads data from the current position of *stream* into the locations given by *argument* (if any).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fseek

int fseek(FILE *_stream_, long _offset_, int _origin_);

Include	STDIO.H
stream	Pointer to target stream
offset	Number of bytes from origin
origin	Initial position: **SEEK_CUR**, **SEEK_END**, **SEEK_SET**
Returns	0 if successful; a nonzero value if not

Moves the file-position indicator associated with *stream* to a new location that is *offset* bytes from *origin*. It has limited use in text mode.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fsetpos

int fsetpos(FILE *_stream_, const fpos_t *_pos_);

Include	STDIO.H, ERRNO.H
stream	Pointer to target stream
pos	Position-indicator storage
Returns	0 if successful; a nonzero value if not
errno	**EBADF**, **EINVAL**

Sets the file-position indicator for *stream* to the value of *pos*.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fsopen

FILE *_fsopen(const char *_filename_, const char *_mode_, int _shflag_)

Include	STDIO.H, SHARE.H
filename	File path name
mode	Access permissions: **"r"**, **"r+"**, **"r+b"**, **"r+t"**, **"w"**, **"w+"**, **"w+b"**, **"w+t"**, **"a"**, **"a+"**, **"a+b"**, **"a+t"**
shflag	**SH_COMPAT**, **SH_DENYRW**, **SH_DENYWR**, **SH_DENYRD**, **SH_DENYNO**
Returns	A pointer to the stream if successful; **NULL** if not

Opens a file as a stream and prepares it for shared reading and writing.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fstat

int fstat(int _handle_, struct stat *_buffer_);

Include	SYS\TYPES.H, SYS\STAT.H, ERRNO.H
handle	Handle of open file
buffer	Pointer to structure to store results
Returns	0 if successful; –1 if not
errno	**EBADF**

Gets information about the open file associated with *handle* and stores it in the structure that *buffer* points to.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_fstrcat

char _far * _far _fstrcat(char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to destination string
string2	Far pointer to source string
Returns	A far pointer to the concatenated string

Appends *string2* to *string1*. Model-independent version of **strcat**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrchr

char _far * _far _fstrchr(const char _far *_string_, int _c_);

Include	STRING.H
string	Far pointer to source string
c	Character to be located
Returns	A far pointer to the first occurrence of *c* in *string*

Searches for the first occurrence of *c* in *string*. Model-independent version of **strchr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrcmp

int _far _fstrcmp(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1, *string2*	Far pointers to strings to compare
Returns	A value less than 0 if *string1* < *string2*; 0 if *string1* = *string2*; a value greater than 0 if *string1* > *string2*

Compares two strings. Model-independent version of **strcmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrcpy

char _far * _far _fstrcpy(char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to destination string
string2	Far pointer to source string
Returns	*string1*

Copies *string2* to *string1*. Model-independent version of **strcpy**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrcspn

size_t _far _fstrcspn(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to source string
string2	Far pointer to character set
Returns	The index of the first character in *string1* that belongs to the set of characters specified by *string2*

Finds first substring in *string1* consisting of characters not in *string2*. Model-independent version of **strcspn**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrdup

char _far * _far _fstrdup(const char _far *_string_);

Include	STRING.H
string	Far pointer to source string
Returns	A far pointer to the storage space containing the duplicate string

Duplicates *string*, copying it to memory allocated by **_fmalloc**. Model-independent version of **strdup**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstricmp

int _far _fstricmp(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1, *string2*	Far pointers to strings to compare
Returns	A value less than 0 if *string1* < *string2*; 0 if *string1* = *string2*; a value greater than 0 if *string1* > *string2*

Compares two strings without regard to case. Model-independent version of **stricmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrlen

size_t _fstrlen(const char _far *_string_);

Include	STRING.H
string	Far pointer to null-terminated string
Returns	The length in bytes of *string*, not including the terminating null character ("\0")

Finds length of string. Model-independent version of **strlen**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrlwr

char _far * _far _fstrlwr(char _far *_string_);

Include	STRING.H
string	Far pointer to string to be converted
Returns	A far pointer to the converted string

Converts any uppercase letters in the given null-terminated *string* to lowercase. Other characters are not affected. Model-independent version of **strlwr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrncat

char _far * _far _fstrncat(char _far *_string1_, const char _far *_string2_, size_t _count_);

Include	STRING.H
string1	Far pointer to destination string
string2	Far pointer to source string
count	Number of characters appended
Returns	A far pointer to the concatenated string

Appends, at most, the first *count* characters of *string2* to *string1* and terminates the resulting string with a null character ('\0'). Model-independent version of **strncat**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrncmp

int _far _fstrncmp(const char _far *_string1_, const char _far *_string2_, size_t _count_);

Include	STRING.H
string1, *string2*	Far pointers to strings to compare
count	Number of characters compared
Returns	A value less than 0 if *string1* < *string2*; 0 if *string1* = *string2*; a value greater than 0 if *string1* > *string2*

Compares, at most, the first *count* characters of *string1* and *string2*. Model-independent version of **strncmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrncpy

char _far * _far _fstrncpy(char _far *_string1_, const char _far *_string2_, size_t _count_);

Include	STRING.H
string1	Far pointer to destination string
string2	Far pointer to source string
count	Number of characters copied
Returns	*string1*

Copies exactly *count* characters of *string2* to *string1*. Model-independent version of **strncpy**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrnicmp

int _far _fstrnicmp(const char _far *_string1_, const char _far *_string2_, size_t _count_);

Include	STRING.H
string1, *string2*	Far pointers to strings to compare
count	Number of characters compared
Returns	A value less than 0 if *string1* < *string2*; 0 if *string1* = *string2*; a value greater than 0 if *string1* > *string2*

Compares, at most, the first *count* characters of *string1* and *string2*, without regard to letter case. Model-independent version of **strnicmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrnset

char _far * _far _fstrnset(char _far *_string_, int _c_, size_t _count_);

Include	STRING.H
string	Far pointer to string to be initialized
c	Character setting
count	Number of characters set
Returns	A far pointer to the altered string

Sets, at most, the first *count* characters of *string* to the character *c*. Model-independent version of **strnset**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrpbrk

char _far * _far _fstrpbrk(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to source string
string2	Far pointer to character set
Returns	A far pointer to the first occurrence of any character from *string2* in *string1*

Finds the first occurrence of a character from one string in another. Model-independent version of **strpbrk**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrrchr

char _far * _far _fstrrchr(const char _far *_string_, int _c_);

Include	STRING.H
string	Far pointer to searched string
c	Character to be located
Returns	A far pointer to the last occurrence of *c* in *string*

Finds the last occurrence of a given character in a string. Model-independent version of **strrchr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrrev

char _far * _far _fstrrev(char _far *_string_);

Include	STRING.H
string	Far pointer to string to be reversed
Returns	A far pointer to the altered string

Reverses the order of the characters in *string*. The terminating null character (**'\0'**) remains in place. Model-independent version of **strrev**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrset

char _far * _far _fstrset(char _far *_string_, int _c_);

Include	STRING.H
string	Far pointer to string to be set
c	Far pointer to character setting
Returns	A far pointer to the altered string

Sets all of the characters of *string* to *c*, except the terminating null character (**'\0'**). Model-independent version of **strset**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrspn

size_t _far _fstrspn(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to searched string
string2	Far pointer to character set
Returns	An integer value specifying the length of the segment in *string1* consisting entirely of characters in *string2*

Returns the index of the first character in *string1* that does not belong to the set of characters specified by *string2*. This value is equivalent to the length of the initial substring of *string1* that consists entirely of characters from *string2*. Model-independent version of **strspn**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrstr

char _far * _far _fstrstr(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to searched string
string2	Far pointer to string to search for
Returns	A far pointer to the first occurrence of *string2* in *string1*

Finds the first occurrence of a given string in another string. Model-independent version of **strstr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrtok

char _far * _far _fstrtok(char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to string containing token(s)
string2	Far pointer to set of delimiter characters
Returns	A far pointer to the first token in *string1*; successive calls against the same string return the subsequent tokens in the string

Reads *string1* as a series of zero or more tokens and *string2* as the set of characters serving as delimiters of the tokens in *string1*. Model-independent version of **strtok**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrupr

char _far * _far _fstrupr(char _far *_string_);

Include	STRING.H
string	Far pointer to string to be capitalized
Returns	A far pointer to the converted string

Converts any lowercase letters in *string* to uppercase. Other characters are not affected. Model-independent version of **strupr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

ftell

long ftell(FILE **stream* **);**

Include	STDIO.H, ERRNO.H
stream	Pointer to target stream
Returns	The current position if successful; –1L if not
errno	**EBADF, EINVAL**

Gets the current position of the file-position indicator associated with *stream*. Use with **fseek**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ftime

void ftime(struct timeb **timeptr* **);**

Include	SYS\TYPES.H, SYS\TIMEB.H
timeptr	Pointer to structure defined in SYS\TIMEB.H
Returns	No return value

Gets the current time and stores it in the structure that *timeptr* points to.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_fullpath

char *_fullpath(char *_buffer_, const char *_pathname_, size_t _maxlen_);

Include	STDLIB.H
buffer	Pointer to user path name buffer
pathname	Pointer to path name fragment
maxlen	Maximum length of *buffer*
Returns	A pointer to the absolute path in *buffer* if successful; NULL if not

Expands the path name fragment *pathname* to form a fully qualified path that is stored in *buffer*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fwrite

size_t fwrite(const void *_buffer_, size_t _size_, size_t _count_, FILE *_stream_);

Include	STDIO.H
buffer	Pointer to data to be written
size	Item size in bytes
count	Maximum number of items to be written
stream	Pointer to target stream
Returns	The number of full items actually written, which may be less than *count* if an error occurs

Writes up to *count* items of length *size* from *buffer* to the output *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

gcvt

char *gcvt(double *value*, **int** *digits*, **char *** *buffer*);

Include	STDLIB.H
value	Value to be converted
digits	Number of significant digits stored
buffer	Storage location for result
Returns	A pointer to the string of digits

Converts a floating-point *value* to a character string and stores the string in *buffer*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_getactivepage

short _far _getactivepage(void);

Include	GRAPH.H
Returns	The number of the current active page

Determines the current active page.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_getarcinfo

short _far _getarcinfo(struct xycoord _far *_start_, struct xycoord _far *_end_,
struct xycoord _far *_fillpoint_);

Include	GRAPH.H
start	Starting point of arc
end	Ending point of arc
fillpoint	Point to begin pie fill
Returns	A nonzero value if successful; 0 if not

Finds the endpoints in viewport coordinates of the arc most recently drawn by **_arc** or **_pie**.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getbkcolor

long _far _getbkcolor(void);

Include	GRAPH.H
Returns	The current background color value

Determines the current background color.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

getc

int getc(FILE *_stream_);

Include	STDIO.H
stream	Pointer to target stream
Returns	The character read if successful; **EOF** to indicate an error or end-of-file

Reads a single character from the current *stream* position and increases the associated file-position indicator to point to the next character.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

getch

int getch(void);

Include CONIO.H

Returns The character read

Reads without echoing a single character from the console.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

getchar

int getchar(void);

Include STDIO.H

Returns The character read if successful; **EOF** to indicate an error or end-of-file

Reads a single character from the current standard input stream, **stdin**, and increases the associated file-position indicator to point to the next character.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

getche

int getche(void);

Include CONIO.H

Returns The character read

Reads a single character from the console and echoes the character read.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_getcolor

short _far _getcolor(void);

Include GRAPH.H

Returns The current color index

Determines the current color.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getcurrentposition

struct xycoord _far _getcurrentposition(void);

Include GRAPH.H

Returns The current position as an **xycoord** structure defined in GRAPH.H

Gets the current graphics output position in view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getcurrentposition_w

struct _wxycoord _far _getcurrentposition_w(void);

Include GRAPH.H

Returns The current position as a **_wxycoord** structure defined in GRAPH.H

Gets the current graphics output position in window coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

getcwd

char *getcwd(char *_buffer_, int _maxlen_);

Include	DIRECT.H, ERRNO.H
buffer	Storage location for path name
maxlen	Maximum length of path name
Returns	*buffer* if successful; **NULL** if not
errno	**ENOMEM, ERANGE**

Gets the full path name (including disk drive specification) of the current working directory and stores it at *buffer*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_getdcwd

char *_getdcwd(int _drive_, char *_buffer_, int _maxlen_);

Include	DIRECT.H, ERRNO.H
drive	Disk drive (A: = 1, B: = 2)
buffer	Storage location for path name
maxlen	Maximum length of path name
Returns	*buffer* if successful; **NULL** if not
errno	**ENOMEM, ERANGE**

Gets the full path name (including disk drive specification) of the current working directory on *drive* and stores it at *buffer*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_getdrive

int _getdrive(void);

Include DIRECT.H

Returns The integer indicating the current working drive (A: = 1, B: = 2)

Gets the current working drive.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

getenv

char *getenv(const char *_varname_);

Include STDLIB.H

varname Name of environment variable

Returns A pointer to the environment table entry containing the current string value of *varname*; **NULL** if the variable is not defined

Searches the list of environment variables for an entry corresponding to *varname*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_getfillmask

unsigned char _far * _far _getfillmask(unsigned char _far *_mask_);

Include GRAPH.H

mask Mask array

Returns **NULL** if no mask is set

Returns the current fill mask used in the **_floodfill**, **_pie**, **_ellipse**, **_polygon**, and **_rectangle** functions.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getfontinfo

short _far _getfontinfo(struct _fontinfo _far *_fontbuffer_);

Include	GRAPH.H
fontbuffer	Font information
Returns	–1 if a font has not been registered or loaded

Returns the current font characteristics in a **_fontinfo** structure defined in GRAPH.H.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getgtextextent

short _far _getgtextextent(unsigned char _far *_text_);

Include	GRAPH.H
text	Text to be analyzed
Returns	The width in pixels; –1 if a font has not been registered

Returns the width required to print the text string in the current font, using **_outgtext**.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getgtextvector

short xycoord _far _getgtextvector(void);

Include	GRAPH.H
Returns	Vector determining the direction of font text rotation as an **xycoord** structure defined in GRAPH.H

Finds the orientation for font text output by **_setgtextvector**.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getimage

void _far _getimage(short *x1*, **short** *y1*, **short** *x2*, **short** *y2*, **char _huge** **image*);

Include	GRAPH.H
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
image	Storage buffer for screen image
Returns	No return value; check **_grstatus**

Stores the screen image defined by a specified bounding rectangle into the buffer *image*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getimage_w

void _far _getimage_w(double *wx1*, **double** *wy1*, **double** *wx2*, **double** *wy2*,
char _huge **image*);

Include	GRAPH.H
wx1, *wy1*	Upper left corner of bounding rectangle
wx2, *wy2*	Lower right corner of bounding rectangle
image	Storage buffer for screen image
Returns	No return value; check **_grstatus**

Stores the screen image defined by a specified bounding rectangle into the buffer *image*. Defines the bounding rectangle with the window coordinates (*wx1*, *wy1*) and (*wx2*, *wy2*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getimage_wxy

void _far _getimage_wxy(struct _wxycoord _far *_pwxy1_,
struct _wxycoord _far *_pwxy2_, char _huge *_image_);

Include	GRAPH.H
pwxy1	Upper left corner of bounding rectangle
pwxy2	Lower right corner of bounding rectangle
image	Storage buffer for screen image
Returns	No return value; check **_grstatus**

Stores the screen image defined by a specified bounding rectangle into the buffer *image*. Defines the bounding rectangle with the window coordinate pairs *pwxy1* and *pwxy2*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getlinestyle

unsigned short _far _getlinestyle(void);

Include	GRAPH.H
Returns	The default mask if no mask has been set

Returns the current line-style mask, which is used by the **_lineto**, **_polygon**, and **_rectangle** functions.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getphyscoord

struct xycoord _far _getphyscoord(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	View coordinate to translate
Returns	The physical coordinates in an **xycoord** structure defined in GRAPH.H

Translates the view coordinate (*x*, *y*) to physical coordinates, returning them in an **xycoord** structure.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

getpid

int getpid(void);

Include	PROCESS.H
Returns	The process identification

Gets the identification number of the calling process.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_getpixel

short _far _getpixel(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	Pixel location
Returns	The color index if successful; –1 if not

Determines the pixel value at the location specified in viewport coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getpixel_w

short _far _getpixel_w(double *wx*, **double** *wy* **);**

Include	GRAPH.H
wx, *wy*	Pixel location
Returns	The color index if successful; –1 if not

Determines the pixel value at the location specified in window coordinates.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

gets

char *gets(char * *buffer* **);**

Include	STDIO.H
buffer	Storage location for input string
Returns	The argument if successful; **NULL** if not, or if end-of-file is reached

Reads a line from the standard input stream, **stdin**, and stores it at *buffer*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_gettextcolor

short _far _gettextcolor(void);

Include	GRAPH.H
Returns	Index of the current text color

Determines the current text attribute.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_gettextcursor

short _far _gettextcursor(void);

Include GRAPH.H

Returns The current cursor attribute if successful; −1 if not

Determines the current cursor shape in text modes.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_gettextposition

struct rccoord _far _gettextposition(void);

Include GRAPH.H

Returns The text position as an **rccoord** structure defined in GRAPH.H

Gets the current text position as an **rccoord** structure.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_gettextwindow

void _far _gettextwindow(short _far *r1, short _far *c1, short _far *r2, short _far *c2);

Include GRAPH.H

r1 Top row of current text window

c1 Leftmost column of current text window

r2 Bottom row of current text window

c2 Rightmost column of current text window

Returns No return value

Gets the boundaries of the current text window.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_getvideoconfig

struct videoconfig _far * _far _getvideoconfig(struct videoconfig _far *_config_);

Include	GRAPH.H
config	Configuration information
Returns	The video configuration information in a **videoconfig** structure defined in GRAPH.H

Gets information about the current graphics environment as a **videoconfig** structure.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_getviewcoord

struct xycoord _far _getviewcoord(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	Physical coordinates to translate
Returns	View coordinates in an **xycoord** structure defined in GRAPH.H

Translates the specified physical coordinates to view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getviewcoord_w

struct xycoord _far _getviewcoord_w(double *wx*, **double** *wy* **);**

Include	GRAPH.H
wx, *wy*	Window coordinates to translate
Returns	View coordinates in an **xycoord** structure defined in GRAPH.H

Translates the specified window coordinates to view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getviewcoord_wxy

struct xycoord _far _getviewcoord_wxy(struct _wxycoord _far *_pwxy1_);

Include	GRAPH.H
pwxy1	Window coordinate to translate
Returns	View coordinates in an **xycoord** structure defined in GRAPH.H

Translates the specified window coordinate structure to view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getvisualpage

short _far _getvisualpage(void);

Include	GRAPH.H
Returns	The number of the current visual page

Gets the current visual page number.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

getw

int getw(FILE *_stream_);

Include	STDIO.H
stream	Pointer to target stream
Returns	The integer value read if successful; **EOF** to indicate failure or end-of-file

Reads the next binary value of type **int** from *stream* and increases the associated file pointer to point to the next unread character.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_getwindowcoord

struct _wxycoord _far _getwindowcoord(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	Physical coordinates to translate
Returns	The coordinates in an **_wxycoord** structure defined in GRAPH.H

Translates the view coordinates (*x*, *y*) to window coordinates and returns them in an **_wxycoord** structure.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getwritemode

short _far _getwritemode(void);

Include	GRAPH.H
Returns	**_GPSET**, **_GXOR**, **_GAND**, **_GOR**, **_GPRESET**; –1 if an error occurs

Returns the current logical write mode used by **_lineto**, **_rectangle**, and **_polygon**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

gmtime

struct tm *gmtime(const time_t * *timer* **);**

Include	TIME.H
timer	Pointer to stored Greenwich mean time value
Returns	A pointer to the structure result

Converts the *timer* value to a structure.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_grstatus

short _far _grstatus(void);

Include	GRAPH.H
Returns	The status of the most recently called graphics function

Determines whether errors or warnings were generated by the most recently called graphics function.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

halloc

void _huge *halloc(long *num*, **size_t** *size* **);**

Include	MALLOC.H
num	Number of elements
size	Length in bytes of each element
Returns	A pointer to the allocated space

Allocates a huge array from the operating system consisting of *num* elements, each of which is *size* bytes long.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_harderr

void _harderr(void(_far *_handler_)( ));

Include	DOS.H
handler	New INT 0x24 handler
Returns	No return value

Establishes the user-defined routine that *handler* points to as the handler for DOS INT 0x24, which is invoked when a hardware error occurs during the execution of an I/O request.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_hardresume

void _hardresume(int *result* **);**

Include	DOS.H
result	**_HARDERR_IGNORE**, **_HARDERR_RETRY**, **_HARDERR_ABORT**, **_HARDERR_FAIL**
Returns	No return value

Allows a user-defined hard I/O error handler to return to DOS.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_hardretn

void _hardretn(int *error* **);**

Include	DOS.H
error	Error to return from
Returns	No return value

Allows a user-defined hard I/O error handler to return directly to the application program.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_heapadd

int _heapadd(void _far *_memblock_, size_t _size_);

Include	MALLOC.H
memblock	Far pointer to heap memory
size	Size in bytes of memory to add
Returns	0 if successful; –1 if not

Adds an unused piece of memory to the heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_heapchk

int _heapchk(void);

Include	MALLOC.H
Returns	**_HEAPOK**, **_HEAPEMPTY**, **_HEAPBADBEGIN**, **_HEAPBADNODE**

Checks the heap for minimal consistency.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_heapmin

int _heapmin(void);

Include	MALLOC.H
Returns	0 if successful; –1 if not

Releases unused memory in the heap back to the operating system.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_heapset

int _heapset(unsigned int *fill* **);**

Include	MALLOC.H
fill	Fill character
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADBEGIN, _HEAPBADNODE**

Checks the heap for minimal consistency and then sets the heap's free entries with the *fill* value.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_heapwalk

int _heapwalk(_HEAPINFO ***entryinfo*);**

Include	MALLOC.H
entryinfo	Structure to contain information about the next heap entry, defined in MALLOC.H
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADPTR, _HEAPBADBEGIN, _HEAPBADNODE, _HEAPEND**

Walks through the heap, one entry per call, returning a pointer to a **_HEAPINFO** structure that contains information about the next heap entry.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

hfree

void hfree(void _huge ***memblock*);**

Include	MALLOC.H
memblock	Pointer to allocated memory block
Returns	No return value

Deallocates a huge memory block obtained with **halloc** and returns the freed memory to the operating system.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

hypot

double hypot(double *x*, **double** *y* **);**

Include	MATH.H, ERRNO.H
x, *y*	Floating-point values
Returns	The length of the hypotenuse if successful; **HUGE_VAL** on overflow
errno	**ERANGE**

Calculates the length of the hypotenuse of a right triangle, given the length of the two sides *x* and *y*.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

hypotl

long double hypotl(long double *x*, **long double** *y* **);**

Include	MATH.H, ERRNO.H
x, *y*	Long double-precision floating-point values
Returns	The length of the hypotenuse if successful; **_LHUGE_VAL** on overflow
errno	**ERANGE**

Calculates the length of the hypotenuse of a right triangle, given the length of the two sides *x* and *y*, using the long double-precision floating-point version of arguments and return values.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_imagesize

long _far _imagesize(short *x1*, **short** *y1*, **short** *x2*, **short** *y2*);

Include	GRAPH.H
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
Returns	The storage size of the image in bytes

Calculates the number of bytes needed to store the image defined by the bounding rectangle specified by the coordinates (*x1*, *y1*) and (*x2*, *y2*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_imagesize_w

long _far _imagesize_w(double *wx1*, **double** *wy1*, **double** *wx2*, **double** *wy2*);

Include	GRAPH.H
wx1, *wy1*	Upper left corner of bounding rectangle
wx2, *wy2*	Lower right corner of bounding rectangle
Returns	The storage size of the image in bytes

Calculates the number of bytes needed to store the image defined by the bounding rectangle specified by the window coordinates (*wx1*, *wy1*) and (*wx2*, *wy2*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_imagesize_wxy

long _far _imagesize_wxy(struct _wxycoord _ far **pwxy1*, **struct _wxycoord _far** **pwxy2*);

Include	GRAPH.H
pwxy1	Upper left corner of bounding rectangle
pwxy2	Lower right corner of bounding rectangle
Returns	The storage size of the image in bytes

Calculates the number of bytes needed to store the image defined by the bounding rectangle specified by the window coordinate pairs *pwxy1* and *pwxy2*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

inp

int inp(unsigned *port* **);**

Include	CONIO.H
port	Port number
Returns	The byte read from *port*

Reads a byte from the specified port. (Note: For protected-mode libraries, use a .DEF file containing this line: SEGMENTS _IOSEG CLASS 'IOSEG_CODE' IOPL.)

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

inpw

unsigned inpw(unsigned *port* **);**

Include	CONIO.H
port	Port number
Returns	The word read from *port*

Reads a word from the specified port. (Note: For protected-mode libraries, use a .DEF file containing this line: SEGMENTS _IOSEG CLASS 'IOSEG_CODE' IOPL.)

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

intdos

int intdos(union REGS ****inregs*, union REGS ****outregs** **);**

Include	DOS.H
inregs	Register values on call
outregs	Register values on return
Returns	The value of the AX register after the system call is completed

Invokes the DOS system call specified by register values defined in *inregs* and returns the effect of the system call in *outregs*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

intdosx

int intdosx(union REGS *_inregs_, union REGS *_outregs_, struct SREGS *_segregs_);

Include	DOS.H
inregs	Register values on call and return
outregs	Register values on return
segregs	Segment-register values on call and on return
Returns	The value of the AX register after the system call is completed

Invokes the DOS system call specified by register values defined in *inregs* and *segregs* and returns the effect of the system call in *outregs* and *segregs*. It accepts segment-register values specifying segments and pointers.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

int86

int int86(int *_intnum_, union REGS *_inregs_, union REGS *_outregs_);

Include	DOS.H
intnum	Interrupt number
inregs	Register values on call
outregs	Register values on return
Returns	The value of the AX register after the interrupt returns

Executes the 8086-processor-family interrupt specified by the interrupt number *intnum*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

int86x

int int86x(int *intnum*, **union REGS** **inregs*, **union REGS** **outregs*,
struct SREGS **segregs*);

Include	DOS.H
intnum	Interrupt number
inregs	Register values on call
outregs	Register values on return
segregs	Segment-register values on call and on return
Returns	The value of the AX register after the interrupt returns

Executes the 8086-processor-family interrupt specified by the interrupt number *intnum*. It accepts segment-register values specifying segments and pointers.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

isalnum

int isalnum(int *c*);

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for alphanumeric character ('A'–'Z', 'a'–'z', or '0'–'9').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isalpha

int isalpha(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for letter ('A'–'Z' or 'a'–'z').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isascii

int isascii(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for ASCII character (0x00 - 0x7f).

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isatty

int isatty(int *handle* **);**

Include	IO.H
handle	Handle referring to device to be tested
Returns	A nonzero value if the device is a character device; 0 if not

Determines whether *handle* is associated with a character device (a terminal, console, printer, or serial port).

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

iscntrl

int iscntrl(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for control character (0x00 – 0x1f or 0x7f).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isdigit

int isdigit(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for digit ('0'–'9').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isgraph

int isgraph(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for graphical characters (any printable character except white space).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

islower

int islower(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for lowercase character ('a'–'z').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isprint

int isprint(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for printable character (0x20 – 0x7e).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ispunct

int ispunct(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for punctuation character.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isspace

int isspace(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for white space character (0x09 – 0x0d or 0x20).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isupper

int isupper(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for uppercase character ('A'–'Z').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isxdigit

int isxdigit(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for hexadecimal digit ('A'–'F','a'–'f', or '0'–'9').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

itoa

char *itoa(int *value*, **char *** *string*, **int** *radix*);

Include	STDLIB.H
value	Number to be converted
string	String result
radix	Base of *value*
Returns	A pointer to *string*

Converts the digits of the given *value* to a null-terminated character string and stores the result (up to 17 bytes) in *string*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

jn

double jn(int *n*, **double** *x*);

Include	MATH.H
n	Integer order
x	Floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order *n*).

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_jnl

long double _jnl(int *n*, **long double** *x* **);**

Include	MATH.H
n	Integer order
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order *n*). Long double-precision floating-point version of **jn**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

j1

double j1(double *x* **);**

Include	MATH.H
x	Floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order 1).

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_j1l

long double _j1l(long double *x* **);**

Include	MATH.H
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order 1). Long double-precision floating-point version of **j1**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

j0

double j0(double *x* **);**

Include	MATH.H
x	Floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order 0).

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_j0l

long double _j0l(long double *x* **);**

Include	MATH.H
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order 0). Long double-precision floating-point version of **j0**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

kbhit

int kbhit(void);

Include	CONIO.H
Returns	A nonzero value if a key has been pressed; 0 if not

Checks the console for an unread keystroke.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

labs

long labs(long *n* **);**

Include	STDLIB.H or MATH.H
n	Long integer value
Returns	The absolute value of *n*

Calculates the long-integer absolute value of *n*.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

ldexp

double ldexp(double *x*, **int** *exp* **);**

Include	MATH.H, ERRNO.H
x	Floating-point value
exp	Integer exponent
Returns	The value of $x * 2^{exp}$ if successful; ± **HUGE_VAL** (depending on the sign of *x*) on overflow
errno	**ERANGE**

Converts the mantissa and exponent to a floating-point value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ldexpl

long double ldexpl(long double *x*, **int** *exp* **);**

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
exp	Integer exponent
Returns	The value of $x * 2^{exp}$ if successful; ±**_LHUGE_VAL** (depending on the sign of *x*) on overflow
errno	**ERANGE**

Converts the mantissa and exponent to a floating-point value. Long double-precision floating-point version of **ldexp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

ldiv

ldiv_t ldiv(long int *numer*, **long int** *denom* **);**

Include	STDLIB.H
numer	Numerator
denom	Denominator
Returns	The quotient and remainder as an **ldiv_t** structure

Computes the quotient and remainder for a pair of long integer values.

■ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

lfind

void *lfind(const void ***_key_, const void ***_base_**, unsigned int ***_num_**, unsigned int** _width_**,**
int(***_compare_)(const void ***_elem1_**, const void ***_elem2_**));**

Include	SEARCH.H
key	Object to search for
base	Pointer to base of search data
num	Number of elements
width	Element width
compare	Pointer to comparison function
elem1	Pointer to the key for the search
elem2	Pointer to the array element to be compared with the key
Returns	A pointer to an array element corresponding to *key* if successful; **NULL** if not

Performs a linear search for the value *key* in an array of *num* elements, each of *width* bytes in size.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_lineto

short _far _lineto(short _x_**, short** _y_**);**

Include	GRAPH.H
x, *y*	End point
Returns	A nonzero value if successful; 0 if not

Draws a line from the current position up to and including the view coordinate (*x*, *y*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_lineto_w

short _far _lineto_w(double *wx*, **double** *wy* **);**

Include	GRAPH.H
wx, *wy*	End point
Returns	A nonzero value if successful; 0 if not

Draws a line from the current position up to and including the window coordinate (*wx*, *wy*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

localeconv

struct lconv *localeconv(void);

Include	LOCALE.H
Returns	A pointer to a structure of type **lconv**

Gets detailed information on locale settings.

■ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

localtime

struct tm *localtime(const time_t * *timer* **);**

Include	TIME.H
timer	Pointer to stored time structure
Returns	Pointer to a structure resolved; **NULL** if time cannot be interpreted

Converts a time stored as a **long** value to a structure of type **tm**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

locking

int locking(int *handle*, **int** *mode*, **long** *nbytes*);

Include	SYS\LOCKING.H, IO.H, ERRNO.H
handle	File handle
mode	**LK_LOCK, LK_RLCK, LK_NBLCK, LK_NBRLCK, LK_UNLCK**
nbytes	Number of bytes to lock
Returns	0 if successful; –1 if not
errno	**EACCESS, EBADF, EDEADLOCK, EINVAL**

Locks or unlocks *nbytes* bytes of the file specified by *handle*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

log

double log(double *x*);

Include	MATH.H, ERRNO.H
x	Floating-point value
Returns	The natural logarithm of *x*
errno	**EDOM, ERANGE**

Calculates the natural logarithm of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

logl

long double logl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
Returns	The natural logarithm of *x*
errno	**EDOM, ERANGE**

Calculates the natural logarithm of *x*. Long double-precision floating-point version of **log**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

log10

double log10(double *x* **);**

Include	MATH.H, ERRNO.H
x	Floating-point value
Returns	The base-10 logarithm of *x*
errno	**EDOM, ERANGE**

Calculates the base-10 logarithm of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

log10l

long double log10l(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
Returns	The base-10 logarithm of *x*
errno	**EDOM, ERANGE**

Calculates the base-10 logarithm of *x*. Long double-precision floating-point version of **log10**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

longjmp

void longjmp(jmp_buf *env*, **int** *value* **);**

Include	SETJMP.H
env	Variable in which environment is stored
value	Value to be returned to **setjmp** call
Returns	No return value

Restores a stack environment and execution locale previously saved in *env* by **setjmp**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_lrotl

unsigned long _lrotl(unsigned long *value*, **int** *shift* **);**

Include	STDLIB.H
value	Value to be rotated
shift	Number of bits to shift
Returns	The rotated value

Rotates the *value* to the left by *shift* bits.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_lrotr

unsigned long _lrotr(unsigned long *value*, **int** *shift* **);**

Include	STDLIB.H
value	Value to be rotated
shift	Number of bits to shift
Returns	The rotated value

Rotates the *value* to the right by *shift* bits.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

lsearch

void *lsearch(const void *_key_, const void *_base_, unsigned int *_num_, unsigned int _width_, int(*_compare_)(const void *_elem1_, const void *_elem2_));

Include	SEARCH.H
key	Object to search for
base	Pointer to base of search data
num	Number of elements
width	Element width
compare	Pointer to comparison function
elem1	Pointer to the key for the search
elem2	Pointer to the array element to be compared with the key
Returns	A pointer to the array element *base* that matches *key* if the key is found; a pointer to the newly added item at the end of the array otherwise

Performs a linear search for the value *key* in an array of *num* elements, each of *width* bytes in size. If the value is not found, it adds *key* to the end of the list.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

lseek

long lseek(int _handle_, long _offset_, int _origin_);

Include	IO.H, STDIO.H, ERRNO.H
handle	Handle referring to open file
offset	Number of bytes from *origin*
origin	**SEEK_SET, SEEK_CUR, SEEK_END**
Returns	The offset in bytes of the new position if successful; –1L if not
errno	**EBADF, EINVAL**

Moves the file pointer associated with *handle* to a new location that is *offset* bytes from *origin*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ltoa

char *ltoa(long *value*, **char** **string*, **int** *radix* **);**

Include	STDLIB.H
value	Number to be converted
string	String result
radix	Base of *value*
Returns	A pointer to *string*

Converts the digits of *value* to a null-terminated character string and stores the result (up to 33 bytes) in *string*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_makepath

void _makepath(char **path*, **char** **drive*, **char** **dir*, **char** **fname*, **char** **ext* **);**

Include	STDLIB.H
path	Full path-name buffer
drive	Drive letter
dir	Directory path
fname	File name
ext	File extension
Returns	No return value

Creates a single path name, composed of a drive letter, directory path, file name, and file-name extension.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

malloc

void *malloc(size_t *size* **);**

Include	STDLIB.H or MALLOC.H
size	Number of bytes to allocate
Returns	A pointer to the allocated space

Allocates a memory block of at least *size* bytes.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

matherr

int matherr(struct exception ***except*);**

Include	MATH.H
except	Math exception information
Returns	A nonzero value if successful; 0 if not

Processes errors generated by the functions of the math library. The user can provide a different **matherr** definition for special treatment of math errors.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_matherrl

int _matherrl(struct _exceptionl ***except*);**

Include	MATH.H
except	Math exception information
Returns	A nonzero value if successful; 0 if not

Processes errors generated by the long double-precision functions of the math library.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

max

type **max**(*type a*, *type b*);

Include	STDLIB.H
type	Any numerical data type
a, *b*	Values to compare
Returns	The larger of the two arguments

Compares two values of the same data type, returning the larger one.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_memavl

size_t _memavl(void);

Include	MALLOC.H
Returns	The size in bytes as an unsigned integer

Returns the approximate amount of memory available for dynamic memory allocation in the near heap (default data segment).

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

memccpy

void *memccpy(void ****dest*, **void ****src***, **int** *c*, **unsigned int** *count*);**

Include	STRING.H or MEMORY.H
dest	Pointer to destination
src	Pointer to source
c	Last character to copy
count	Number of characters
Returns	A pointer to the byte in *dest* that immediately follows the copied character (*c*) if successful; NULL if *count* bytes copied before *c* was found

Copies zero or more bytes of the source to the destination, up to and including the first occurrence of the character *c* or until *count* bytes have been copied, whichever comes first.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

memchr

void *memchr(const void ****buf*, **int** *c*, **size_t** *count*);**

Include	STRING.H or MEMORY.H
buf	Pointer to buffer
c	Character to search for
count	Maximum number of characters to scan
Returns	A pointer to the first location of *c* in *buf* if successful; NULL if not

Looks for the first occurrence of a character in the first *count* bytes of the buffer.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

memcmp

int memcmp(const void *_buf1_, const void *_buf2_, size_t _count_);

Include	STRING.H or MEMORY.H
buf1, *buf2*	Pointers to buffers to compare
count	Maximum number of characters to compare
Returns	A value less than, equal to, or greater than 0 if the size of the buffer pointed to by *buf1* is less than, equal to, or greater than the size of the buffer pointed to by *buf2*.

Compares the first *count* bytes of two buffers and returns a value indicating their relationship.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

memcpy

void *memcpy(void *_dest_, const void *_src_, size_t _count_);

Include	STRING.H or MEMORY.H
dest	Pointer to destination buffer
src	Pointer to the buffer to copy from
count	Number of characters to copy
Returns	*dest*

Copies *count* bytes of *src* to *dest*. The first *count* bytes of *src* and *dest* should not overlap.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

memicmp

int memicmp(void *_buf1_, void *_buf2_, unsigned int _count_);

Include	STRING.H or MEMORY.H
buf1, *buf2*	Pointers to buffers to compare
count	Number of characters
Returns	A value less than, equal to, or greater than 0 if the size of the buffer pointed to by *buf1* is less than, equal to, or greater than the size of the buffer pointed to by *buf2*.

Compares the first *count* bytes of *buf1* and *buf2* byte-by-byte, without regarding case (all letters are considered lowercase), and returns a value indicating their relationship.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_memmax

size_t _memmax(void);

Include	MALLOC.H
Returns	The size of the largest contiguous block of memory that can be allocated from the near heap if successful; 0 if nothing more can be allocated from the near heap

Determines the size of the largest contiguous free memory block in the near heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

memmove

void *memmove(void *_dest_, const void *_src_, size_t _count_);

Include	STRING.H
dest	Pointer to destination buffer
src	Pointer to source buffer
count	Number of characters to copy
Returns	The value of *dest*

Copies *count* characters from *src* to *dest* and handles overlap properly.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

memset

void *memset(void *_dest_, int _c_, size_t _count_);

Include	STRING.H or MEMORY.H
dest	Pointer to destination buffer
c	Character to set to
count	Number of characters to be set
Returns	A pointer to *dest*

Sets the first *count* bytes of *dest* to the character *c*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

min

type **min(** *type a*, *type b* **);**

Include	STDLIB.H
type	Any numeric data type
a, *b*	Values to compare
Returns	The smaller of the two arguments

Compares two values of the same data type and returns the smaller of them.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

mkdir

int mkdir(char ******dirname* **);**

Include	DIRECT.H, ERRNO.H
dirname	Path name for new directory
Returns	0 if successful; –1 if not
errno	**EACCESS, ENOENT**

Creates a new directory with the specified directory name.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

mktemp

char *mktemp(char ****template* **);

Include	IO.H
template	File-name pattern
Returns	A pointer to the modified template if successful; **NULL** if not

Creates a unique file name by modifying the given file-name pattern.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

mktime

time_t mktime(struct tm **timeptr* **);**

Include	TIME.H
timeptr	Local time structure
Returns	The calendar time encoded as a type **time_t** value if successful; −1 cast as type **time_t** if not

Converts the local time to a calendar value.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

modf

double modf(double *x*, **double** **intptr* **);**

Include	MATH.H
x	Floating-point value
intptr	Pointer to stored integer portion
Returns	The signed fractional portion of *x*

Breaks down the floating-point value *x* into fractional and integer parts with the same sign as *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

modfl

long double modfl(long double *x*, **long double ****intptr* **);**

Include	MATH.H
x	Long double-precision floating-point value
intptr	Pointer to stored integer portion
Returns	The signed fractional portion of *x*

Breaks down the long double-precision floating-point value of *x* into fractional and integer parts with the same sign as *x*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

movedata

void movedata(unsigned int *srcseg*, **unsigned int** *srcoff*, **unsigned int** *destseg*, **unsigned int** *destoff*, **unsigned int** *count* **);**

Include	STRING.H or MEMORY.H
srcseg	Segment address of source
srcoff	Segment offset of source
destseg	Segment address of destination
destoff	Segment offset of destination
count	Number of bytes
Returns	No return value

Copies *count* bytes from the source address specified by *srcseg*:*srcoff* to the destination address specified by *destseg*:*destoff*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_moveto

struct xycoord _far _moveto(short *x*, **short** *y*);

Include	GRAPH.H
x, *y*	Target position
Returns	The coordinates of the previous position in an **xycoord** structure

Moves the current graphics-output position to the specified view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_moveto_w

struct _wxycoord _far _moveto_w(double *wx*, **double** *wy*);

Include	GRAPH.H
wx, *wy*	Target position
Returns	The coordinates of the previous position in a **_wxycoord** structure

Moves the current graphics-output position to the specified window coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_msize

size_t _msize(void **memblock*);

Include	MALLOC.H
memblock	Pointer to memory block
Returns	The size in bytes

An unsigned integer representing the size in bytes of the memory block allocated by a call to **calloc**, **malloc**, or **realloc**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_ncalloc

void _near *_ncalloc(size_t *num*, size_t *size*);

Include	MALLOC.H
num	Number of elements
size	Length in bytes of each element
Returns	A near pointer to the allocated space if successful; **NULL** if there is insufficient memory available

Allocates storage for an array of *num* elements, each of length *size* bytes in the near heap. Storage is initialized to zero.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_nexpand

void _near *_nexpand(void _near **memblock*, size_t *size*);

Include	MALLOC.H
memblock	Pointer to previously allocated memory block
size	New size in bytes
Returns	A near pointer to the resized block if successful; **NULL** if there is insufficient memory

Changes the size of a previously allocated memory block by attempting to expand or contract the block without moving its location in the near heap.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_nfree

void _nfree(void _near *_memblock_);

Include	MALLOC.H
memblock	Allocated memory block in the near heap
Returns	No return value

Frees a memory block in the near heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nheapchk

int _nheapchk(void);

Include	MALLOC.H
Returns	**_HEAPOK**, **_HEAPEMPTY**, **_HEAPBADBEGIN**, **_HEAPBADNODE**

Checks for minimal consistency on the dynamically allocated memory space in the default data segment, or "near heap." The consistency check determines whether all the near heap entries are within the bounds of the heap's current memory allocation.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nheapmin

int _nheapmin(void);

Include	MALLOC.H
Returns	0 if successful; –1 if not

Releases unused memory in the near heap back to the operating system.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nheapset

int _nheapset(unsigned int *fill*);

Include	MALLOC.H
fill	Fill character
Returns	**_HEAPOK**, **_HEAPEMPTY**, **_HEAPBADBEGIN**, **_HEAPBADNODE**

Checks for minimal consistency in the near heap and then sets the heap's free entries with the *fill* value.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nheapwalk

int _nheapwalk(_HEAPINFO *_entryinfo_);

Include	MALLOC.H
entryinfo	Structure to contain information about the next near heap entry defined in MALLOC.H
Returns	**_HEAPOK**, **_HEAPEMPTY**, **_HEAPBADPTR**, **_HEAPBADBEGIN**, **_HEAPBADNODE**, **_HEAPEND**

Walks through the near heap, one entry per call, returning a pointer to a **_HEAPINFO** structure that contains information about the next near heap entry.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nmalloc

void _near *_nmalloc(size_t *size*);

Include	MALLOC.H
size	Number of bytes to allocate
Returns	A near pointer to the allocated space if successful; **NULL** if not

Allocates a memory block of at least *size* bytes in the near heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nmsize

size_t _nmsize(void _near *_memblock_);

Include	MALLOC.H
memblock	Pointer to near memory block
Returns	The size in bytes

An unsigned integer representing the size in bytes of the near-heap memory block allocated by a call to **_ncalloc**, **_nmalloc**, or **_nrealloc**

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nrealloc

void _near *_nrealloc(void _near *_memblock_, size_t _size_);

Include	MALLOC.H
memblock	Pointer to previously allocated memory block
size	New size in bytes
Returns	A near pointer to the reallocated memory block if successful; **NULL** if not

Changes the size and possibly the location of a memory block in the near heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nstrdup

char _near * _far _nstrdup(const char _far *_string_);

Include	STRING.H
string	Source string
Returns	A near pointer to the storage space containing the duplicate string

Duplicates *string*, copying it to memory allocated by **_nmalloc**. Model-independent form of **strdup**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

onexit

onexit_t onexit(onexit_t *func* **);**

Include	STDLIB.H
func	Function to call on exit
Returns	A pointer to the function if successful; **NULL** if there is no space left to store the function pointer

Creates a register of functions to be called (last-in, first-out) when the program terminates normally.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

open

int open(char **filename*, **int** *oflag* [[, **int** *pmode*]]);

Include	SYS\TYPES.H, SYS\STAT.H, IO.H, ERRNO.H, FCNTL.H
filename	File path name
oflag	**O_APPEND, O_BINARY, O_CREAT, O_EXCL, O_RDONLY, O_RDWR, O_TEXT, O_TRUNC, O_WRONLY**
pmode	**S_IWRITE, S_IREAD, S_IREAD | S_IWRITE**
Returns	A handle for the opened file if successful; −1 if not
errno	**EACCESS, EEXIST, EINVAL, EMFILE, ENOENT**

Opens the file specified by *filename* and prepares the file for subsequent reading or writing, as defined by *oflag* and, optionally, *pmode*.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_outgtext

void _far _outgtext(unsigned char _far * *text* **);**

Include	GRAPH.H
text	Text to be output
Return	No return value

Outputs the null-terminated string *text* using the current font and rotation on the screen at the current graphics-output position.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_outmem

void _far _outmem(unsigned char _far * *text*, **short** *length* **);**

Include	GRAPH.H
text	Text to be output
length	Number of characters to be output
Returns	No return value

Outputs the specified number of characters from a buffer.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

outp

int outp(unsigned *port*, **int** *databyte* **);**

Include	CONIO.H
port	Port number
databyte	Output value
Returns	The output data

Writes a byte to the specified output port. (Note: For protected-mode libraries, use a .DEF file containing this line: SEGMENTS _IOSEG CLASS, 'IOSEG_CODE' IOPL.)

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

outpw

unsigned outpw(unsigned *port*, **unsigned** *dataword* **);**

Include	CONIO.H
port	Port number
dataword	Output value
Returns	The output data

Writes a word to the specified output port. (Note: For protected-mode libraries, use a .DEF file containing this line: SEGMENTS _IOSEG CLASS 'IOSEG_CODE' IOPL.)

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_outtext

void _far _outtext(unsigned char _far **text* **);**

Include	GRAPH.H
text	Text to be output
Return	No return value

Outputs the null-terminated string, *text*, at the current text position using the current text color.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_pclose

int _pclose(FILE **stream* **);**

Include	STDIO.H
stream	Return value from previous call to **_popen**
Returns	The exit status of the child command if successful; -1 if not

Closes *stream* and waits for the associated child command.

□ ANSI □ DOS ■ OS/2 ■ UNIX ■ XENIX

perror

void perror(const char *string* **);**

Include	STDIO.H
string	User-supplied message
Returns	No return value

Prints an error message to **stderr**. The *string* argument is printed first, followed by a colon, then by the system error message for the last library call that produced the error, and finally by a newline character.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_pg_analyzechart

short _far _pg_analyzechart(chartenv _far *env*, **char _far * _far** *categories*, **float _far** *values*, **short** *n* **);**

Include	PGCHART.H
env	Chart environment variable
categories	Array of category variables
values	Array of data values
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Analyzes a single series of data and fills the chart environment with default values for a single-series bar, column, or line chart, depending on the type specified in the function call.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_pg_analyzechartms

short _far _pg_analyzechartms(chartenv _far * *env*, **char _far * _far *** *categories*,
float _far * *values*, **short** *nseries*, **short** *n*, **short** *arraydim*,
char _far * _far * *serieslabels*);

Include	PGCHART.H
env	Chart environment variable
categories	Array of category variables
values	Array of data values
nseries	Number of series to chart
n	Number of data values to chart
arraydim	Row dimension of data array
serieslabels	Array of labels for series
Returns	0 if successful; a nonzero value if not

Analyzes a multiple series of data and fills the chart environment with default values for a multiseries bar, column, or line chart, depending on which type is specified in the function call.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_analyzepie

short _far _pg_analyzepie(chartenv _far *_env_, char _far * _far *_categories_,
float _far *_values_, short _far *_explode_, short _n_);

Include	PGCHART.H
env	Chart environment structure
categories	Array of category labels
values	Array of data values
explode	Array of explode flags
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Analyzes a single series of data and fills the chart environment for a pie chart using the data contained in the array *values*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_analyzescatter

short _far _pg_analyzescatter(chartenv _far *_env_, float _far *_xvalues_,
float _far *_yvalues_, short _n_);

Include	PGCHART.H
env	Chart environment structure
xvalues	Array of *x*-axis data values
yvalues	Array of *y*-axis data values
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Analyzes a single series of data and fills the chart environment for a single-series scatter diagram.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_analyzescatterms

short _far _pg_analyzescatterms(chartenv _far *_env_, float _far *_xvalues_,
float _far *_yvalues_, short _nseries_, short _n_, short _rowdim_,
char _far * _far *_serieslabels_);

Include	PGCHART.H
env	Chart environment structure
xvalues	Array of *x*-axis data values
yvalues	Array of *y*-axis data values
nseries	Number of series to chart
n	Number of data values to chart
rowdim	Row dimension of data array
serieslabels	Array of labels for series
Returns	0 if successful; a nonzero value if not

Analyzes a multiple series of data and fills the chart environment for a multiseries scatter diagram.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_chart

short _far _pg_chart(chartenv _far *_env_, char _far * _far *_categories_,
float _far *_values_, short _n_);

Include	PGCHART.H
env	Chart environment variable
categories	Array of category variables
values	Array of data values
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Displays a single-series bar, column, or line chart, depending on the type specified in the chart environment variable.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_chartms

short _far _pg_chartms(chartenv _far *_env_, char _far * _far *_categories_,
float _far *_values_, short _nseries_, short _n_, short _arraydim_,
char _far * _far *_serieslabels_);

Include	PGCHART.H
env	Chart environment variable
categories	Array of category variables
values	Array of data values
nseries	Number of series to chart
n	Number of data values to chart
arraydim	Row dimension of data array
serieslabels	Array of labels for series
Returns	0 if successful; a nonzero value if not

Displays a multiseries bar, column, or line chart, depending on the type specified in the chart environment.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_chartpie

short _far _pg_chartpie(chartenv _far *_env_, char _far * _far *_categories_,
float _far *_values_, short _far *_explode_, short _n_);

Include	PGCHART.H
env	Chart environment structure
categories	Array of category labels
values	Array of data values
explode	Array of explode flags
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Displays a pie chart for the data contained in the array *values*. Pie charts are formed from a single series of data (there is no multiseries version of pie charts as there is for other chart types).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_chartscatter

short _far _pg_chartscatter(chartenv _far *_env_, float _far *_xvalues_,
float _far *_yvalues_, short _n_);

Include	PGCHART.H
env	Chart environment structure
xvalues	Array of *x*-axis data values
yvalues	Array of *y*-axis data values
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Displays a scatter diagram for a single series of data.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_chartscatterms

short _far _pg_chartscatterms(chartenv _far *_env_, float _far *_xvalues_,
float _far *_yvalues_, short _nseries_, short _n_, short _rowdim_,
char _far * _far *_serieslabels_);

Include	PGCHART.H
env	Chart environment structure
xvalues	Array of *x*-axis data values
yvalues	Array of *y*-axis data values
nseries	Number of series to chart
n	Number of data values to chart
rowdim	Row dimension of data array
serieslabels	Array of labels for series
Returns	0 if successful; a nonzero value if not

Displays a scatter diagram for more than one series of data.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_defaultchart

short _far _pg_defaultchart(chartenv _far *_env_, short _charttype_, short _chartstyle_);

Include	PGCHART.H
env	Chart environment structure
charttype	**_PG_BARCHART**, **_PG_COLUMNCHART**, **_PG_LINECHART**, **_PG_SCATTERCHART**, **_PG_PIECHART**
chartstyle	Chart style 1 or 2
Returns	0 if successful; a nonzero value if not

Initializes all necessary variables in the chart environment for the chart type by the variable *charttype*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_getchardef

short _far _pg_getchardef(short *charnum*, **unsigned char _far ***_chardef_ **);**

Include	PGCHART.H
charnum	ASCII number of character
chardef	Pointer to 8-by-8 bit map array
Returns	0 if successful; a nonzero value if not

Retrieves the current 8-by-8 pixel bit map for the character with the ASCII number *charnum*. The bit map is stored in the *chardef* array.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_getpalette

short _far _pg_getpalette(paletteentry _far ***_palette_ **);

Include	PGCHART.H
palette	Pointer to first palette structure in array
Returns	0 if successful; **_BADSCREENMODE** if current palettes have not been initialized by a previous call to **_pg_setpalette**

Retrieves palette colors, line styles, fill patterns, and plot characters for all palettes. The pointer *palette* points to an array of palette structures that will contain the desired palette values.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_getstyleset

void _far _pg_getstyleset(unsigned short _far ***_styleset_ **);

Include	PGCHART.H
styleset	Pointer to current styleset
Return	No return value

Retrieves the contents of the current styleset.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_hlabelchart

short _far _pg_hlabelchart(chartenv _far *_env_, short _x_, short _y_, short _color_,
char _far *_label_);

Include	PGCHART.H
env	Chart environment structure
x	Pixel *x*-coordinate for text
y	Pixel *y*-coordinate for text
color	Color code for text
label	Label text
Returns	0 if successful; a nonzero value if not

Writes text horizontally on the screen. The arguments *x* and *y* are pixel coordinates for the beginning location of text relative to the upper left corner of the chart window.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_initchart

short _far _pg_initchart(void);

Include	PGCHART.H
Returns	0 if successful; a nonzero value if not

Initializes the presentation graphics package. It initializes the color and style pools, resets the chartline styleset, builds default palette modes, and reads the presentation graphics character font definition from the disk. This function is required in all programs that use presentation graphics.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_resetpalette

short _far _pg_resetpalette(void);

Include PGCHART.H

Returns 0 if successful; **_BADSCREENMODE** if the screen mode is not valid

Sets the palette colors, line styles, fill patterns, and plot characters for the palette to the default for the current screen mode.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_resetstyleset

void _far _pg_resetstyleset(void);

Include PGCHART.H

Returns No return value

Reinitializes the styleset to the default values for the current screen mode.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_setchardef

short _far _pg_setchardef(short *charnum*, **unsigned char _far ***_chardef_ **);**

Include PGCHART.H

charnum ASCII number of character

chardef Pointer to 8-by-8 bit map array

Returns No return value

Sets the 8-by-8 pixel bit map for the character with the ASCII number *charnum*. The bit map is stored in the *chardef* array.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_setpalette

short _far _pg_setpalette(paletteentry _far *_palette_);

Include	PGCHART.H
palette	Pointer to first palette structure in array
Returns	0 if successful; **_BADSCREENMODE** if the new palettes are not valid

Sets palette colors, line styles, fill patterns, and plot characters for all palettes. The pointer *palette* points to an array of palette structures that contains the desired palette values.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_setstyleset

void _far _pg_setstyleset(unsigned short _far *_styleset_);

Include	PGCHART.H
styleset	Pointer to new styleset
Returns	No return value

Sets the current styleset.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_vlabelchart

short _far _pg_vlabelchart(chartenv _far *_env_, short _x_, short _y_, short _color_, char _far *_label_);

Include	PGCHART.H
env	Chart environment structure
x	Pixel *x*-coordinate for text
y	Pixel *y*-coordinate for text
color	Color code for text
label	Label text
Returns	0 if successful; a nonzero value if not

Writes text vertically on the screen. The arguments *x* and *y* are pixel coordinates for the beginning location of text relative to the upper left corner of the chart window.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pie

short _far _pie(short _control_, short _x1_, short _y1_, short _x2_, short _y2_, short _x3_, short _y3_, short _x4_, short _y4_);

Include	GRAPH.H
control	**_GBORDER**, **_GFILLINTERIOR**
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
x3, *y3*	Start vector
x4, *y4*	End vector
Returns	A nonzero value if successful; 0 if not

Draws a pie-shaped wedge by drawing an elliptical arc whose center and two endpoints are joined by lines. Points are defined using the view coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pie_w

short _far _pie_w(short *control*, **double** *x1*, **double** *y1*, **double** *x2*, **double** *y2*, **double** *x3*, **double** *y3*, **double** *x4*, **double** *y4*);

Include	GRAPH.H
control	Fill-control constant
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
x3, *y3*	Start vector
x4, *y4*	End vector
Returns	A nonzero value if successful; 0 if not

Draws a pie-shaped wedge by drawing an elliptical arc whose center and two endpoints are joined by lines. Points are defined using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pie_wxy

short _far _pie_wxy(short *control*, **struct _wxycoord _far** **pwxy1*, **struct _wxycoord _far** **pwxy2*, **struct _wxycoord _far** **pwxy3*, **struct _wxycoord _far** **pwxy4*);

Include	GRAPH.H
control	**_GBORDER**, **_GFILLINTERIOR**
pwxy1	Upper left corner of bounding rectangle
pwxy2	Lower right corner of bounding rectangle
pwxy3	Start vector
pwxy4	End vector
Returns	A nonzero value if successful; 0 if not

Draws a pie-shaped wedge by drawing an elliptical arc whose center and two endpoints are joined by lines. Points are defined using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pipe

int _pipe(int ***phandles*****, unsigned int** ***psize*****, int** ***textmode*** **);**

Include	IO.H, FCNTL.H, ERRNO.H
phandles	Pointer to read and write handles (*phandles*[0] and *phandles*[1], respectively)
psize	Amount of memory, in bytes, reserved for the pipe
textmode	**O_BINARY**, **O_TEXT**
Returns	0 if successful; –1 if not
errno	**EMFILE**, **ENFILE**

Creates a pipe for both reading and writing, generally in preparation for linking it to a child process.

☐ ANSI ☐ DOS ■ OS/2 ■ UNIX ■ XENIX

_polygon

short _far _polygon(short ***control*****, struct xycoord _far** ***points*****, short** ***numpoints*** **);**

Include	GRAPH.H
control	**_GFILLINTERIOR**, **_GBORDER**
points	Array of **xycoord** structures specifying the polygon's vertices
numpoints	Number of vertices
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws or scan-fills a polygon, using the view coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_polygon_w

short _far _polygon_w(short *control*, **double _far** ******points*, **short** *numpoints*);

Include	GRAPH.H
control	**_GFILLINTERIOR**, **_GBORDER**
points	Values specifying the polygon's vertices
numpoints	Number of vertices
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws or scan-fills a polygon, using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_polygon_wxy

short _far _polygon_wxy(short *control*, **struct _wxycoord _far** ******points*,
short *numpoints*);

Include	GRAPH.H
control	**_GFILLINTERIOR**, **_GBORDER**
points	Array of **_wxycoord** structures specifying the polygon's vertices
numpoints	Number of vertices
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws or scan-fills a polygon, using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_popen

FILE * _popen(char *_command_, char *_mode_);

Include	STDIO.H
command	Command string
mode	Access permissions: **"r"**, **"rt"**, **"rb"**, **"w"**, **"wt"**, **"wb"**
Returns	A stream associated with one end of the pipe if successful; **NULL** if not

Creates a pipe and asynchronously executes a child copy of the command processor.

□ ANSI □ DOS ■ OS/2 ■ UNIX ■ XENIX

pow

double pow(double x**, double** y **);**

Include	MATH.H, ERRNO.H
x	Number to be raised
y	Power of x
Returns	The value of x^y; 1 if x is not 0 and y is 0; ± **HUGE_VAL** if x is 0 and y is negative; 0 if both x and y are 0 or if x is negative and y is not an integer
errno	**EDOM**, **ERANGE**

Computes the value of x^y.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

powl

long double powl(long double *x*, **long double** *y* **);**

Include	MATH.H, ERRNO.H
x	Number to be raised
y	Power of *x*
Returns	The value of x^y: 1 if *x* is not 0 and *y* is 0; ± **_LHUGE_VAL** if *x* is 0 and *y* is negative; 0 if both *x* and *y* are 0 or if *x* is negative and *y* is not an integer
errno	**EDOM, ERANGE**

Computes the value of x^y. Long double-precision floating-point version of **pow**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

printf

int printf(const char ******format* **[[,** *argument* **]]...);**

Include	STDIO.H
format	Format-control string
argument	Optional arguments, variables of the type specified in the format-control string
Returns	The number of characters printed if successful; a negative value if not

Formats and prints a series of characters and values to the standard output stream, **stdout**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

putc

int putc(int *c*, **FILE** **stream* **);**

Include	STDIO.H
c	Character to be written
stream	Pointer to target stream
Returns	The character written if successful; **EOF** may indicate failure

Writes the single character *c* to the output *stream* at the current position.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

putch

int putch(int *c* **);**

Include	CONIO.H
c	Character to be output
Returns	*c* if successful; **EOF** if not

Writes the character *c* directly to the console.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

putchar

int putchar(int *c* **);**

Include	STDIO.H
c	Character to be written
Returns	The character written if successful; **EOF** may indicate failure

Writes the single character *c* to the standard output stream, **stdout**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

putenv

int putenv(char **envstring* **);**

Include	STDLIB.H
envstring	Environment-string definition
Returns	0 if successful; –1 if not

Adds or removes an environment variable or modifies the value of an existing one.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_putimage

void _far _putimage(short *x*, **short** *y*, **char _huge ****image*, **short** *action* **);**

Include	GRAPH.H
x, *y*	Position of upper left corner of image
image	Stored image buffer
action	**_GAND**, **_GOR**, **_GPRESET**, **_GPSET**, **_GXOR**
Returns	No return value; check **_grstatus**

Transfers to the screen the image stored in the buffer *image*, placing the upper left corner of the image at the view coordinate (*x*, *y*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_putimage_w

void _far _putimage_w(double *wx*, **double** *wy*, **char _huge *****image*, **short** *action* **);**

Include	GRAPH.H
wx, *wy*	Position of upper left corner of image
image	Stored image buffer
action	**_GAND**, **_GOR**, **_GPRESET**, **_GPSET**, **_GXOR**
Returns	No return value; check **_grstatus**

Transfers to the screen the image stored in the buffer *image*, placing the upper left corner of the image at the window coordinate (*wx*, *wy*).

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

puts

int puts(const char *****string* **);

Include	STDIO.H
string	String to be output
Returns	A nonnegative value if successful; **EOF** if not

Writes *string* to the standard output stream, **stdout**, replacing the string's terminating null character (**'\0'**) with a newline character (**\n**) in the output stream.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

putw

int putw(int *binint*, **FILE ***stream*);**

Include	STDIO.H
binint	Binary integer to be output
stream	Pointer to target stream
Returns	The value written if successful; **EOF** may indicate an error

Writes a binary value of type **int** to the current position of the *stream*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

qsort

void qsort(void ***base*, size_t *num*, **size_t** *width*,
int(***compare*)(const void ***elem1*, const void ***elem2*));

Include	STDLIB.H, SEARCH.H
base	Start of array
num	Number of elements in array
width	Element size in bytes
compare	Comparison function
elem1	Pointer to key for the search
elem2	Pointer to the array element to be compared with the key
Returns	No return value

Implements a quick-sort algorithm to sort an array of *num* elements, each of *width* bytes.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

raise

int raise(int *sig* **);**

Include	SIGNAL.H
sig	**SIGABRT, SIGFPE, SIGILL, SIGINT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGUSR3, SIGBREAK**
Returns	0 if successful; a nonzero value if not

Raises the *sig* signal condition in the executing program.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

rand

int rand(void);

Include	STDLIB.H
Returns	A pseudorandom integer in the range 0 to **RAND_MAX**

Generates a pseudorandom number.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

read

int read(int *handle*, **void ****buffer*, **unsigned int** *count* **);**

Include	IO.H, ERRNO.H
handle	Handle referring to open file
buffer	Storage location for data
count	Maximum number of bytes
Returns	The number of bytes read if successful; −1 if not
errno	**EBADF**

Attempts to read *count* bytes into *buffer* from the file associated with *handle*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

realloc

void *realloc(void *_memblock_, size_t _size_);

Include	STDLIB.H or MALLOC.H
memblock	Pointer to previously allocated memory block
size	New size in bytes
Returns	A **void** pointer to the reallocated memory block if successful; **NULL** if not

Changes the size and, possibly, the location of a previously allocated memory block.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_rectangle

short _far _rectangle(short *control*, **short** *x1*, **short** *y1*, **short** *x2*, **short** *y2* **);**

Include	GRAPH.H
control	**_GFILLINTERIOR**, **_GBORDER**
x1, *y1*	Upper left corner
x2, *y2*	Lower right corner
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws a rectangle with the current color, line style, and write mode, using the view coordinate system.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_rectangle_w

short _far _rectangle_w(short *control*, **double** *wx1*, **double** *wy1*, **double** *wx2*, **double** *wy2*);

Include	GRAPH.H
control	**_GFILLINTERIOR**, **_GBORDER**
wx1, *wy1*	Upper left corner
wx2, *wy2*	Lower right corner
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws a rectangle with the current color, line style, and write mode, using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_rectangle_wxy

short _far _rectangle_wxy(short *control*, **struct _wxycoord _far** **pwxy1*, **struct _wxycoord _far** **pwxy2*);

Include	GRAPH.H
control	**_GFILLINTERIOR**, **_GBORDER**
pwxy1	Upper left corner
pwxy2	Lower right corner
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws a rectangle with the current color, line style, and write mode, using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_registerfonts

short _far _registerfonts(unsigned char _far *_pathname_);

Include	GRAPH.H
pathname	Path name specifying .FON files to be registered
Returns	The number of fonts registered if successful; a negative value if not

Initializes the font graphics system. Font files must be registered with the **_registerfonts** function before any other font-related library function can be used.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_remapallpalette

short _far _remapallpalette(long _far *_colors_);

Include	GRAPH.H
colors	Array of color values
Returns	A nonzero value if successful; 0 if not

Remaps all of the available colors simultaneously.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_remappalette

long _far _remappalette(short _index_, long _color_);

Include	GRAPH.H
index	Color index to redefine
color	Color value to assign to *index*
Returns	The previous color value of *index* if successful; −1 if not

Remaps the specified color index to *color*, which must be a color value supported by the current video configuration.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

remove

int remove(const char *_filename_);

Include	STDIO.H or IO.H, ERRNO.H
filename	Path name of file to be removed
Returns	0 if successful; a nonzero value if not
errno	**EACCESS, ENOENT**

Deletes the file specified by *filename*.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

rename

int rename(const char *_oldname_, const char *_newname_);

Include	STDIO.H or IO.H, ERRNO.H
oldname	Pointer to old name
newname	Pointer to new name
Returns	0 if successful; a nonzero value if not
errno	**EACCESS, ENOENT, EXDEV**

Renames the file or directory specified by *oldname* to the name given by *newname*.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

rewind

void rewind(FILE *_stream_);

Include	STDIO.H
stream	Pointer to target stream
Returns	No return value

Repositions the file pointer associated with *stream* to the beginning of the file.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

rmdir

int rmdir(char *_dirname_);

Include	DIRECT.H, ERRNO.H
dirname	Path name of directory to be removed
Returns	0 if successful; a nonzero value if not
errno	**EACCESS, ENOENT**

Deletes the directory specified by *dirname*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

rmtmp

int rmtmp(void);

Include	STDIO.H
Returns	The number of temporary files closed and deleted

Cleans up all the temporary files in the current directory. The function removes only those files created by **tmpfile** and should be used only in the same directory in which the temporary files were created.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_rotl

unsigned int _rotl(unsigned int *value*, **int** *shift* **);**

Include	STDLIB.H
value	Value to be rotated
shift	Number of bits to shift
Returns	*value* rotated to the left by *shift* bits

Rotates bits to the left.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_rotr

unsigned int _rotr(unsigned int *value*, **int** *shift* **);**

Include	STDLIB.H
value	Value to be rotated
shift	Number of bits to shift
Returns	*value* rotated to the right by *shift* bits

Rotates bits to the right.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

scanf

int scanf(const char **format* [[, *argument*]]... **);**

Include	STDIO.H
format	Format-control string
argument	Optional arguments; pointers to variables of the types specified in the format-control string
Returns	The number of fields successfully converted and assigned; **EOF** if input failure occurs before any conversion

Reads data from the standard input stream, **stdin**, into the locations given by an argument.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_scrolltextwindow

void _far _scrolltextwindow(short *lines* **);**

Include	GRAPH.H
lines	Number of lines to scroll
Returns	No return value

Scrolls the current text window. A positive value for *lines* scrolls the window up, and a negative value scrolls down.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_searchenv

void _searchenv(char ***filename*, char** ****varname*, char** ****pathname*);**

Include	STDLIB.H
filename	Name of file to search for
varname	Environment to search
pathname	Buffer to store complete path
Returns	No return value

Searches for the target file in the specified *varname* domain.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

segread

void segread(struct SREGS ****segregs*);**

Include	DOS.H
segregs	Segment-register values
Returns	No return value

Fills the structure pointed to by *segregs* with the current contents of the segment registers.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_selectpalette

short _far _selectpalette(short *number* **);**

Include	GRAPH.H
number	Palette number
Returns	The previous palette number; –1 if an error occurs

Selects one of several predefined palettes, depending on video mode and hardware (**_MRES4COLOR**, **_MRESNOCOLOR**, and **_ORESCOLOR** modes only).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setactivepage

short _far _setactivepage(short *page* **);**

Include	GRAPH.H
page	Memory page number
Returns	The number of the previous active page if successful; –1 if an error occurs

Specifies the area in memory where output is written for hardware and mode configurations with enough memory to support multiple-screen pages. Note that OS/2 supports only one page. Page 0 is always supported.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_setbkcolor

long _far _setbkcolor(long *color* **);**

Include	GRAPH.H
color	Desired background color
Returns	The previous background color

Sets the current background color to *color*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

setbuf

void setbuf(FILE ***stream*, char** ****buffer*** **);**

Include	STDIO.H
stream	Pointer to target stream
buffer	User-allocated buffer of size **BUFSIZ** or **NULL** for no buffering
Returns	No return value

Allows the user to assign a buffer or disable buffering for *stream*. The argument *stream* must refer to an open file before it has been read or written.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_setcliprgn

void _far _setcliprgn(short *x1*, **short** *y1*, **short** *x2*, **short** *y2* **);**

Include	GRAPH.H
x1, *y1*	Upper left corner of clip region
x2, *y2*	Lower right corner of clip region
Returns	No return value; check **_grstatus**

Limits the display of subsequent graphics output and font text output to the part that fits within a particular area of the screen, known as the clipping region.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_setcolor

short _far _setcolor(short *color* **);**

Include	GRAPH.H
color	Desired color index
Returns	The previous color index if successful; –1 if not

Sets the current color index (used by graphics and font text output) to *color*.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_setfillmask

void _far _setfillmask(unsigned char _far *_mask_);

Include	GRAPH.H
mask	Mask array
Returns	No return value

Sets the current fill mask. The mask is an 8-by-8 array of bits, where each bit represents a pixel. A 1 bit sets the corresponding pixel to the current color, whereas a 0 bit leaves the pixel unchanged.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setfont

short _far _setfont(unsigned char _far *_options_);

Include	GRAPH.H
options	String describing font characteristics
Returns	The font index number if successful; a negative value if not

Finds a single font from the set of registered fonts that has the characteristics specified by the *options* string and returns the index of that font.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setgtextvector

struct xycoord _far _setgtextvector(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	Values defining a vector that determines the direction of font text rotation
Returns	The previous vector in an **xycoord** structure defined in GRAPH.H

Sets the current orientation for font text output. (0, 0) is ignored.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

setjmp

int setjmp(jmp_buf *env* **);**

Include	SETJMP.H
env	Variable in which environment is stored
Returns	0 after saving the stack environment; the value argument of **longjmp** if **setjmp** returns as a result of a **longjmp** call; 1 if the **longjmp** argument is 0

Saves a stack environment that can subsequently be restored using **longjmp**. Used together this way, **setjmp** and **longjmp** provide a way to execute a nonlocal goto.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_setlinestyle

void _far _setlinestyle(unsigned short *mask* **);**

Include	GRAPH.H
mask	Desired line-style mask
Returns	No return value

Selects the mask used for line drawing. The mask is a 16-bit number, where each bit represents a pixel in the line being drawn. If a bit is 1, the corresponding pixel is set to the current color. If a bit is 0, the corresponding pixel is left unchanged.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

setlocale

char *setlocale(int *category*, **const char *** *locale* **);**

Include	LOCALE.H
category	**LC_ALL, LC_COLLATE, LC_TYPE, LC_MONETARY, LC_NUMERIC, LC_TIME, NULL**
locale	"C"
Returns	A pointer to the string associated with *category* for the new locale, if valid; a null pointer if invalid

Sets categories specified for a certain location.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

setmode

int setmode(int *handle*, **int** *mode* **);**

Include	FCNTL.H, IO.H, ERRNO.H
handle	File handle
mode	**O_TEXT, O_BINARY**
Returns	The previous translation mode if successful; –1 if not
errno	**EBADF, EINVAL**

Sets the translation mode of the file given by *handle* to *mode*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_setpixel

short _far _setpixel(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	Target pixel
Returns	The previous value of the target pixel if successful; –1 if not

Sets a pixel at the specified view coordinate location to the current color.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setpixel_w

short _far _setpixel_w(double *wx*, **double** *wy* **);**

Include	GRAPH.H
wx, *wy*	Target pixel
Returns	The previous value of the target pixel if successful; –1 if not

Sets a pixel at the specified window coordinate location to the current color.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_settextcolor

short _far _settextcolor(short *index* **);**

Include	GRAPH.H
index	Desired color index
Returns	The color index of the previous text attribute

Sets the current text attribute to the color index specified by *index*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_settextcursor

short _far _settextcursor(short *attr* **);**

Include	GRAPH.H
attr	0x0707, 0x0007, 0x0607, 0x2000, etc.
Returns	The previous cursor attribute if successful; –1 if not

In text modes, sets the BIOS cursor attribute (the shape) to the value specified by *attr*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_settextposition

struct rccoord _far _settextposition(short *row*, **short** *column* **);**

Include	GRAPH.H
row, *column*	New text output start position
Returns	The previous text position

Sets the current text position to the location (*row*, *column*) within the current text window.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_settextrows

short _far _settextrows(short *rows* **);**

Include	GRAPH.H
rows	Number of text rows requested
Returns	The number of rows actually set if successful; 0 if an unexpected error occurs

Requests the number of text rows to be used in the current video mode.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_settextwindow

void _far _settextwindow(short *r1*, **short** *c1*, **short** *r2*, **short** *c2*);

Include	GRAPH.H
r1, *c1*	Upper left corner of window
r2, *c2*	Lower right corner of window
Returns	No return value; check **_grstatus**

Specifies a window in row and column coordinates where all the text output to the screen is confined.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

setvbuf

int setvbuf(FILE **stream*, **char** **buffer*, **int** *mode*, **size_t** *size*);

Include	STDIO.H
stream	Pointer to target stream
buffer	User-supplied buffer or **NULL**
mode	**_IOFBF**, **_IOLBF**, **_IONBF**
size	Size of buffer
Returns	0 if successful; a nonzero value if not

Allows the user to control both buffering and buffer size for *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_setvideomode

short _far _setvideomode(short *mode* **);**

Include	GRAPH.H
mode	**_MAXRESMODE, _MAXCOLORMODE, _DEFAULTMODE, _TEXTBW40, _TEXTC40, _TEXTBW80, _TEXTC80, _MRES4COLOR, _MRESNOCOLOR, _HRESBW, _TEXTMONO, _HERCMONO, _MRES16COLOR, _HRES16COLOR, _ERESNOCOLOR, _ERESCOLOR, _VRES2COLOR, _VRES16COLOR, _MRES256COLOR, _ORESCOLOR**
Returns	The number of text rows if successful; 0 if not

Selects a screen mode appropriate for a particular hardware/display configuration. Only text modes are available in OS/2.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_setvideomoderows

short _far _setvideomoderows(short *mode*, **short** *rows* **);**

Include	GRAPH.H
mode	**_MAXRESMODE, _MAXCOLORMODE, _DEFAULTMODE, _TEXTBW40, _TEXTC40, _TEXTBW80, _TEXTC80, _MRES4COLOR, _MRESNOCOLOR, _HRESBW, _TEXTMONO, _HERCMONO, _MRES16COLOR, _HRES16COLOR, _ERESNOCOLOR, _ERESCOLOR, _VRES2COLOR, _VRES16COLOR, _MRES256COLOR, _ORESCOLOR**
rows	Number of text rows requested
Returns	The number of rows actually set if successful; 0 if not

Selects a screen mode for a particular hardware/display combination, and requests the number of text rows to be used. Only text modes are available in OS/2.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_setvieworg

struct xycoord _far _setvieworg(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	New origin point
Returns	The physical coordinates of the previous view origin

Moves the viewport origin (0, 0) to the physical point (*x*, *y*). (All other view coordinate points move the same direction and distance.)

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setviewport

void _far _setviewport(short *x1*, **short** *y1*, **short** *x2*, **short** *y2* **);**

Include	GRAPH.H
x1, *y1*	Upper left corner of viewport
x2, *y2*	Lower right corner of viewport
Returns	No return value; check **_grstatus**

Redefines the graphics viewport (the active drawing area within the boundaries of the physical screen).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setvisualpage

short _far _setvisualpage(short *page* **);**

Include	GRAPH.H
page	Visual page number
Returns	The number of the previous visual page if successful; a negative value if not

Specifies the area in memory where output is displayed for hardware and mode configurations with enough memory to support multiple-screen pages. Note that OS/2 supports only one page. Page 0 is always supported.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_setwindow

short _far _setwindow(short *finvert*, **double** *wx1*, **double** *wy1*,
double *wx2*, **double** *wy2* **);**

Include	GRAPH.H
finvert	Invert flag
wx1, *wy1*	Upper left corner of window
wx2, *wy2*	Lower right corner of window
Returns	A nonzero value if successful; 0 if not

Defines a virtual window bounded by the specified coordinates within the current viewport. All *w* and *wxy* drawing functions produce output relative to this window.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setwritemode

short _far _setwritemode(short *action* **);**

Include	GRAPH.H
action	**_GPSET, _GPRESET, _GAND, _GOR, _GXOR**
Returns	The previous write mode; –1 if an error occurs

Sets the current logical write mode used when drawing lines with the **_lineto**, **_rectangle**, and **_polygon** functions.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

signal

void(*signal(int *sig*, **void(*** *func* **)(int** *sig* **[], int** *subcode***[])))(int** *sig* **);**

Include	SIGNAL.H, ERRNO.H
sig	**SIGABRT, SIGBREAK, SIGFPE, SIGILL, SIGINT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGUSR3**
func	Function address or one of **SIG_IGN, SIG_DFL, SIG_SGE, SIG_ACK**
subcode	Optional subcode to the signal number
Returns	The previous value of *func* associated with the given signal if successful; –1 if not
errno	**EINVAL**

Allows a process to choose one of several ways to handle an interrupt signal from the operating system.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sin

double sin(double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The sine of *x*
errno	**ERANGE**

Calculates the sine of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sinh

double sinh(double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The hyperbolic sine of *x*; ± **HUGE_VAL** if not
errno	**ERANGE**

Calculates the hyperbolic sine of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sinhl

long double sinhl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The hyperbolic sine of *x* if successful; ± **_LHUGE_VAL** if not
errno	**ERANGE**

Calculates the hyperbolic sine of *x*. Long double-precision floating-point version of **sinh**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

sinl

long double sinl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The sine of *x*
errno	**ERANGE**

Calculates the sine of *x*. Long double-precision floating-point version of **sin**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

sopen

int sopen(char ***filename*, int** *oflag*, **int** *shflag* [[, **int** *pmode*]] **);**

Include	SYS\TYPES.H, SYS\STAT.H, FCNTL.H, SHARE.H, IO.H
filename	File path name
oflag	**O_APPEND, O_BINARY, O_CREAT, O_EXCL, O_RDONLY, O_RDWR, O_TEXT, O_TRUNC, O_WRONLY**
shflag	**SH_COMPAT, SH_DENYRW, SH_DENYWR, SH_DENYRD, SH_DENYNO**
pmode	**S_IWRITE, S_IREAD, S_IREAD | S_IWRITE**
Returns	A file handle for the opened file if successful; −1 if not
errno	**EACCES, EEXIST, EINVAL, EMFILE, ENOENT**

Opens the file specified by *filename* and prepares the file for subsequent shared reading or writing, as defined by *oflag*, *shflag*, and, optionally, *pmode*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnl

int spawnl(int *mode*, **char** **cmdname*, **char** **arg0*, **char** **arg1*, ... **char** **argn*, **NULL**);

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT**, **P_NOWAIT**, **P_NOWAITO**, **P_OVERLAY**, **P_DETACH**
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG**, **EINVAL**, **ENOENT**, **ENOEXEC**, **ENOMEM**

Creates and executes a new child process, and passes a copy of the current process's environment to the child. Command-line arguments are passed individually to the function.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnle

int spawnle(int *mode*, **char** **cmdname*, **char** **arg0*, **char** **arg1*, ... **char** **argn*, **NULL**, **char** ***envp*);

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT**, **P_NOWAIT**, **P_NOWAITO**, **P_OVERLAY**, **P_DETACH**
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
envp	Array of pointers to environment settings
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG**, **EINVAL**, **ENOENT**, **ENOEXEC**, **ENOMEM**

Creates and executes a new child process. Command-line arguments are passed individually to the function. The child process is passed an array of pointers to environment strings instead of a copy of the current process's environment strings.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnlp

int spawnlp(int *mode*, **char** **cmdname*, **char** **arg0*, **char** **arg1*, ... **char** **argn*, **NULL**);

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT**, **P_NOWAIT**, **P_NOWAITO**, **P_OVERLAY**, **P_DETACH**
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG**, **EINVAL**, **ENOENT**, **ENOEXEC**, **ENOMEM**

Creates and executes a new child process, and passes to the child a copy of the environment of the current process. Command-line arguments are passed individually to the function. The PATH environment variable is used to find the file to be executed if a relative path name is given.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

spawnlpe

int spawnlpe(int *mode*, **char** **cmdname*, **char** **arg0*, **char** **arg1*, ... **char** **argn*, **NULL**, **char** ***envp*);

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT**, **P_NOWAIT**, **P_NOWAITO**, **P_OVERLAY**, **P_DETACH**
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
envp	Array of pointers to environment settings
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG**, **EINVAL**, **ENOENT**, **ENOEXEC**, **ENOMEM**

Creates and executes a new child process. Command-line arguments are passed individually to the function. The PATH environment variable is used to find the file to be executed if it is specified by a relative path name. The child process is passed an array of pointers to environment strings, instead of a copy of the environment strings of the current process.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnv

int spawnv(int *mode*, **char** **cmdname*, **char** ***argv*);

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT**, **P_NOWAIT**, **P_NOWAITO**, **P_OVERLAY**, **P_DETACH**
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG**, **EINVAL**, **ENOENT**, **ENOEXEC**, **ENOMEM**

Creates and executes a new child process, and passes a copy of the current process's environment to the child. Command-line arguments are passed to the function as an array of pointers.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnve

int spawnve(int *mode*, **char** **cmdname*, **char** ***argv* , **char** ***envp*);

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
envp	Array of pointers to environment settings
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process. Command-line arguments are passed to the function as an array of pointers. The child process is passed an array of pointers to environment strings, instead of a copy of the environment strings of the current process.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnvp

int spawnvp(int *mode*, **char** **cmdname*, **char** ***argv*);

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process, and passes a copy of the current process's environment to the child. Command-line arguments are passed to the function as an array of pointers. The PATH environment variable is used to find the file to be executed if necessary and if the file is specified by a relative path name.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnvpe

int spawnvpe(int *mode*, **char** **cmdname*, **char** ***argv*, **char** ***envp*);

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
envp	Array of pointers to environment settings
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process. Command-line arguments are passed to the function as an array of pointers. The PATH environment variable is used to find the file to be executed. It explicitly passes to the child process the strings, instead of a copy of the current process's environment strings array of pointers to environment.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_splitpath

void _splitpath(char **path*, **char** **drive*, **char** **dir*, **char** **fname*, **char** **ext*);

Include	STDLIB.H
path	Full path-name buffer
drive	Drive letter
dir	Directory path
fname	File name
ext	File extension
Returns	No return value

Breaks the full path-name buffer *path* into its four components.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

sprintf

int sprintf(char **buffer*, **const char** **format* [], *argument*[]... **);**

Include	STDIO.H
buffer	Storage location for output
format	Format-control string
argument	Optional arguments; pointers to variables of the types specified in the format-control string
Returns	The number of characters stored in *buffer*, not counting the terminating null character ('\0')

Formats and stores a series of characters and values in *buffer*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sqrt

double sqrt(double *x* **);**

Include	MATH.H, ERRNO.H
x	Nonnegative floating-point value
Returns	The square root of *x* if successful; 0 if not
errno	**EDOM**

Calculates the square root.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sqrtl

long double sqrtl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Nonnegative long double-precision floating-point value
Returns	The square root of *x* if successful; 0 if not
errno	**EDOM**

Calculates the square root. Long double-precision floating-point version of **sqrt**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

srand

void srand(unsigned int *seed* **);**

Include	STDLIB.H
seed	Seed for random-number generation
Returns	No return value

Sets the starting point for generating a series of pseudorandom integers.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sscanf

int sscanf(const char *_buffer_, const char *_format_ [[, _argument_]]...);

Include	STDIO.H
buffer	Stored data
format	Format-control string
argument	Optional arguments; pointers to variables of types specified in the format-control string
Returns	The number of fields that were successfully converted and assigned

Reads data from *buffer* into the locations given by each argument. Every argument must be a pointer to a variable with a type that corresponds to a type specifier in *format*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

stackavail

size_t stackavail(void);

Include	MALLOC.H
Returns	The size in bytes as an unsigned integer

Gets the approximate size in bytes of the stack space available for dynamic memory allocation with **alloca**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

stat

int stat(char **pathname*, **struct stat** **buffer* **);**

Include	SYS\STAT.H, SYS\TYPES.H, ERRNO.H
pathname	Path name of existing file
buffer	Pointer to structure to receive results
Returns	0 if successful; a nonzero value if not
errno	**ENOENT**

Gets information about the file or directory specified by *pathname* and stores it in the structure that *buffer* points to.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_status87

unsigned int _status87(void);

Include	FLOAT.H
Returns	A floating-point status word whose bits are defined in FLOAT.H

Gets the current 8087 floating-point status word.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strcat

char *strcat(char **string1*, **const char** **string2* **);**

Include	STRING.H
string1	Destination string
string2	Source string
Returns	A pointer to the beginning of the concatenated string

Appends *string2* to *string1*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strchr

char *strchr(const char *_string_, int _c_);

Include	STRING.H
string	Source string
c	Character to be located
Returns	A pointer to the first occurrence of *c* in *string*

Searches for the first occurrence of *c* in *string*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strcmp

int strcmp(const char *_string1_, const char *_string2_);

Include	STRING.H
string1, *string2*	Strings to compare
Returns	A value less than, equal to, or greater than 0, depending on whether the string pointed to by *string1* is less than, equal to, or greater than the string pointed to by *string2*

Compares two strings.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strcoll

int strcoll(const char *_string1_, const char *_string2_);

Include	STRING.H
string1, *string2*	Strings to compare
Returns	A value less than, equal to, or greater than 0, depending on whether the string pointed to by *string1* is less than, equal to, or greater than the string pointed to by *string2*

Compares two strings for locale-specific collating sequences.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strcpy

char *strcpy(char *_string1_, const char *_string2_);

Include	STRING.H
string1	Destination string
string2	Source string
Returns	*string1*

Copies *string2* to *string1*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strcspn

size_t strcspn(const char *_string1_, const char *_string2_);

Include	STRING.H
string1	Source string
string2	Character set
Returns	The length of the maximum initial segment of *string1* that consists of characters not in *string2*

Finds first substring in *string1* of characters not in *string2*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_strdate

char *_strdate(char *_datestr_);

Include	TIME.H
datestr	Current date
Returns	A pointer to the date

Copies the current date to the buffer that *datestr* points to, formatted as mm/dd/yy.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strdup

char *strdup(const char *string* **);**

Include	STRING.H
string	Source string
Returns	A pointer to the storage space containing the duplicate string

Duplicates *string*, copying it to memory allocated by **malloc**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

strerror

char *strerror(int *errnum* **);**

Include	STRING.H
errnum	Error number
Returns	A pointer to the error-message string

Maps *errnum* to an error-message string, returning a pointer to the string.

■ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_strerror

char *_strerror(char *string* **);**

Include	STRING.H
string	User-supplied message
Returns	No return value

Returns a pointer to a string containing, in order, the string message, a colon, a space, the system error message for the last library call producing an error, and a newline character. The supplied string message can be a maximum of 94 bytes long.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

strftime

size_t strftime(char **string*, **size_t** *maxsize*, **const char** **format*,
const struct tm **timeptr* **);**

Include	TIME.H
string	Output string
maxsize	Maximum length of string
format	Format-control string
timeptr	**tm** data structure
Returns	The number of characters placed in *string* if the total number, including the terminating null character (**'\0'**), is less than *maxsize*; 0 otherwise

Copies text into a string related to date and time values from a **tm** structure and specified by *format*.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

stricmp

int stricmp(const char **string1*, **const char** **string2* **);**

Include	STRING.H
string1, *string2*	Strings to compare
Returns	A value less than, equal to, or greater than 0, depending on whether the string pointed to by *string1* is less than, equal to, or greater than the string pointed to by *string2*

Compares two strings without regard to case.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strlen

size_t strlen(const char *_string_);

Include	STRING.H
string	Null-terminated string
Returns	The length in bytes of *string*, not including the terminating null character (**'\0'**)

Gets the length of a string.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strlwr

char *strlwr(char *_string_);

Include	STRING.H
string	String to be converted
Returns	A pointer to the converted string

Converts any uppercase letters in the given null-terminated *string* to lowercase. Other characters are not affected.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strncat

char *strncat(char *_string1_, const char *_string2_, size_t _count_);

Include	STRING.H
string1	Destination string
string2	Source string
count	Maximum number of characters to append
Returns	A pointer to the concatenated string

Appends, at most, the first *count* characters of *string2* to *string1* and terminates the resulting string with a null character ('\0').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strncmp

int strncmp(const char *_string1_, const char *_string2_, size_t _count_);

Include	STRING.H
string1, *string2*	Strings to compare
count	Maximum number of characters to compare
Returns	A value less than, equal to, or greater than 0, depending on whether the string pointed to by *string1* is less than, equal to, or greater than the string pointed to by *string2*

Compares, at most, the first *count* characters of *string1* and *string2*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strncpy

char *strncpy(char *_string1_, const char *_string2_, size_t _count_);

Include	STRING.H
string1	Destination string
string2	Source string
count	Number of characters copied
Returns	*string1*

Copies, at most, *count* characters of *string2* to *string1*. If *string2* is shorter than *count*, null ('\0') characters are appended to *string1*. Warning: If the sum of the lengths of *string1* and *string2* is longer than *count*, no null character will be appended.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strnicmp

int strnicmp(const char *_string1_, const char *_string2_, size_t _count_);

Include	STRING.H
string1, *string2*	Strings to compare
count	Maximum number of characters compared
Returns	A value less than, equal to, or greater than 0, depending on whether the string pointed to by *string1* is less than, equal to, or greater than the string pointed to by *string2*

Compares, at most, the first *count* characters of *string1* and *string2*, without regard to letter case.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strnset

char *strnset(char *_string_, int _c_, size_t _count_);

Include	STRING.H
string	String to be initialized
c	Character setting
count	Maximum number of characters set
Returns	A pointer to the altered string

Sets, at most, the first *count* characters of *string* to the character *c*. If *count* is longer than *string*, the length of *string* replaces *count*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

strpbrk

char *strpbrk(const char *_string1_, const char *_string2_);

Include	STRING.H
string1	Source string
string2	Character set
Returns	A pointer to the first occurrence of any character from *string2* in *string1*

Finds the first occurrence in *string1* of any character from *string2*, not including the terminating null character ('\0').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strrchr

char *strrchr(const char *_string_, int _c_);

Include	STRING.H
string	String to be searched
c	Character to be located
Returns	A pointer to the last occurrence of *c* in *string*; a null pointer if *c* is not found

Finds the last occurrence of the character *c* in *string*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strrev

char *strrev(char *_string_);

Include	STRING.H
string	String to be reversed
Returns	A pointer to the altered string

Reverses the order of the characters in *string*. The terminating null character (**'\0'**) remains in place.

□ ANSI ■ DOS ■ OS/2 ❒ UNIX □ XENIX

strset

char *strset(char *_string_, int _c_);

Include	STRING.H
string	String to be set
c	Character setting
Returns	A pointer to the altered string

Sets all of the characters of *string* to *c*, except the terminating null character (**'\0'**).

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strspn

size_t strspn(const char *_string1_, const char *_string2_);

Include	STRING.H
string1	Searched string
string2	Character set
Returns	The length of the initial substring of *string1* that consists entirely of characters from *string2*

Finds the first substring and returns the index of the first character in *string1* that does not belong to the set of characters specified by *string2*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strstr

char *strstr(const char *_string1_, const char *_string2_);

Include	STRING.H
string1	Searched string
string2	String to search for
Returns	A pointer to the first occurrence of *string2* in *string1*; **NULL** if *string2* is not found

Finds a substring.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_strtime

char *_strtime(char *_timestr_);

Include	TIME.H
timestr	Time string
Returns	A formatted string

Copies the current time into the buffer that *timestr* points to, formatted as hh:mm:ss.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strtod

double strtod(const char *_nptr_, char **_endptr_);

Include	STDLIB.H, ERRNO.H
nptr	String to convert
endptr	Pointer to the character causing the end of scan
Returns	The value of the floating-point number if successful; ± **HUGE_VAL** on overflow; 0 otherwise
errno	**ERANGE**

Converts a character string to a double-precision value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strtok

char *strtok(char *_string1_, const char *_string2_);

Include	STRING.H
string1	String containing token(s)
string2	Set of delimiter characters
Returns	A pointer to the first token in *string1*; successive calls against the same string return the subsequent tokens in the string

Reads *string1* as a series of zero or more tokens and *string2* as the set of characters serving as delimiters of the tokens in *string1*. Inserts a null character each time it is called, modifying *string1*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strtol

long strtol(const char **nptr*, **char ******endptr*, **int** *base* **);**

Include	STDLIB.H, ERRNO.H
nptr	String to convert
endptr	Pointer to the character causing the end of scan
base	Number base to use
Returns	The value represented in the string if successful; **LONG_MAX** or **LONG_MIN** on overflow; 0 otherwise
errno	**ERANGE**

Converts a character string to a long-integer value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_strtold

long double _strtold(const char **nptr*, **char ******endptr* **);**

Include	STDLIB.H, ERRNO.H
nptr	String to convert
endptr	Pointer to the character causing the end of scan
Returns	The long double-precision value of the floating-point number if successful; ± **_LHUGE_VAL** on overflow; 0 otherwise
errno	**ERANGE**

Converts a character string to a long double-precision floating-point value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strtoul

unsigned long strtoul(const char *_nptr_, char **_endptr_, int _base_);

Include	STDLIB.H, ERRNO.H
nptr	String to convert
endptr	Pointer to the character causing the end of scan
base	Number base to use
Returns	The converted value if successful; **ULONG_MAX** on overflow; 0 otherwise
errno	**ERANGE**

Converts a character string to an unsigned-long-integer value.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strupr

char *strupr(char *_string_);

Include	STRING.H
string	String to be capitalized
Returns	A pointer to the converted string

Converts any lowercase letters in *string* to uppercase. Other characters are not affected.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strxfrm

size_t strxfrm(char **string1*, **const char** **string2*, **size_t** *count* **);**

Include	STRING.H
string1	Result string
string2	Source string
count	Number of characters transformed
Returns	The length of the transformed string, excluding the terminating null character ('\0').

Transforms the string pointed to by *string2* into a new form, based on locale-specific information and stores the result in *string1*.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

swab

void swab(char **source*, **char** **dest*, **int** *n* **);**

Include	STDLIB.H
source	Data to be copied and swapped
dest	Storage location for swapped data
n	Number of bytes copied
Returns	No return value

Copies *n* bytes from *source*, swaps each pair of adjacent bytes, and stores the result at *dest*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

system

int system(const char *_command_);

Include	PROCESS.H, STDLIB.H, ERRNO.H
command	Command to be executed
Returns	A nonzero value if *command* is **NULL** and the command interpreter is found; otherwise, the return code from spawning the command interpreter
errno	**E2BIG, ENOENT, ENOEXEC, ENOMEM**

Passes *command* to the command interpreter and executes the string as an operating system command.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tan

double tan(double _x_);

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The tangent of *x* if successful; 0 if not
errno	**ERANGE**

Calculates the tangent of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tanh

double tanh(double *x* **);**

Include	MATH.H
x	Angle in radians
Returns	The hyperbolic tangent of *x*

Calculates the hyperbolic tangent of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tanhl

long double tanhl(long double *x* **);**

Include	MATH.H
x	Angle in radians
Returns	The hyperbolic tangent of *x*

Calculates the hyperbolic tangent of *x*. Long double-precision floating-point version of **tanh**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

tanl

long double tanl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The tangent of *x* if successful; 0 if not
errno	**ERANGE**

Calculates the tangent of *x*. Long double-precision floating-point version of **tan**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

tell

long tell(int *handle* **);**

Include	IO.H, ERRNO.H
handle	Handle referring to open file
Returns	The current position of the file pointer (if any) associated with *handle* if successful; −1L if not
errno	**EBADF**

Gets the current position of any file pointer associated with *handle*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

tempnam

char *tempnam(char ****dir*, char ****prefix* **);**

Include	STDIO.H
dir	Target directory if TMP environment variable not defined
prefix	File-name prefix
Returns	A pointer to the name generated if successful; **NULL** if not

Creates a temporary file in another directory.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

time

time_t time(time_t *_timer_);

Include	TIME.H
timer	Storage location for time
Returns	The time in elapsed seconds

Returns the number of seconds elapsed since 00:00:00 Greenwich mean time (GMT), January 1, 1970, according to the system clock.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tmpfile

FILE *tmpfile(void);

Include	STDIO.H
Returns	A stream pointer if successful; a null pointer if not

Creates a temporary file and returns a stream pointer to that file.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tmpnam

char *tmpnam(char *_string_);

Include	STDIO.H
string	Pointer to temporary name
Returns	A pointer to the name generated if successful; NULL if not

Generates a string that can be used as the name of a temporary file.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

toascii

int toascii(int *c* **);**

Include	CTYPE.H
c	Character to be converted
Returns	The converted character

Sets all but the low-order 7 bits of *c* to 0, so that the converted value represents a character in the ASCII character set. If *c* already represents an ASCII character, *c* is unchanged.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tolower

int tolower(int *c* **);**

Include	CTYPE.H
c	Character to be converted
Returns	The lowercase character corresponding to *c*

Converts *c* to lowercase if *c* represents an uppercase letter. Otherwise, *c* is unchanged.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_tolower

int _tolower(int *c* **);**

Include	CTYPE.H
c	Character to be converted
Returns	The lowercase character corresponding to *c*

Converts *c* to lowercase. To be used only if *c* is known to be an uppercase letter. The result is undefined if *c* is not an uppercase letter.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

toupper

int toupper(int *c* **);**

Include	CTYPE.H
c	Character to be converted
Returns	The uppercase character corresponding to *c*

Converts *c* to uppercase if *c* represents a lowercase letter. Otherwise, *c* is unchanged.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_toupper

int _toupper(int *c* **);**

Include	CTYPE.H
c	Character to be converted
Returns	The uppercase character corresponding to *c*

Converts *c* to uppercase. To be used only if *c* is known to be a lowercase letter. The result is undefined if *c* is not a lowercase letter.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tzset

void tzset(void);

Include	TIME.H
Returns	No return value

Using the current setting of the environment variable TZ, assigns values to the global variables **daylight**, **timezone**, and **tzname**, which are used by the **ftime** and **localtime** functions to make corrections from GMT to local time, and by **time** to compute GMT from system time.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ultoa

char *ultoa(unsigned long *value*, **char *** *string*, **int** *radix* **);**

Include	STDLIB.H
value	Number to be converted
string	String result
radix	Base of *value*
Returns	A pointer to *string*

Converts the digits of *value* to a null-terminated character string and stores the result (up to 33 bytes) in *string*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

umask

int umask(int *pmode* **);**

Include	SYS\TYPES.H, SYS\STAT.H, IO.H
pmode	**S_IWRITE**, **S_IREAD**
Returns	The previous value of *pmode*

Sets the file-permission mask of the current process to the mode specified by *pmode*. The file-permission mask is used to modify the permission setting of new files created by **creat**, **open**, or **sopen**.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ungetc

int ungetc(int *c*, **FILE** **stream* **);**

Include	STDIO.H
c	Character to be pushed
stream	Pointer to target stream
Returns	The character argument *c* if successful; **EOF** if not

Pushes the character *c* back onto the input *stream* and clears the end-of-file indicator. A subsequent read operation on the stream begins with *c*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ungetch

int ungetch(int *c* **);**

Include	CONIO.H
c	Character to be pushed
Returns	The character argument *c* if successful; **EOF** if not

Pushes the character *c* back to the console, causing *c* to be the next character read by **getch** or **getche**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

unlink

int unlink(const char **filename* **);**

Include	IO.H or STDIO.H, ERRNO.H
filename	Path name of file to be removed
Returns	0 if successful; –1 if not
errno	**EACCESS, ENOENT**

Deletes the file specified by *filename*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_unregisterfonts

void _far _unregisterfonts(void);

Include	GRAPH.H
Returns	No return value

Frees memory previously allocated and used by the **_registerfonts** function. The **_unregisterfonts** function removes the header information for all fonts and unloads the currently selected font data from memory.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

utime

int utime(char *_filename_, struct utimbuf *_times_);

Include	SYS\TYPES.H, SYS\UTIME.H, ERRNO.H
filename	File path name
times	Pointer to stored time values
Returns	0 if successful; –1 if not
errno	**EACCESS, EINVAL, EMFILE, ENOENT**

Sets the modification time for the file specified by *filename*.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

va_arg

type **va_arg(** **va_list** *arg_ptr*, *type* **);**

Include	STDARG.H (ANSI) or VARARGS.H (UNIX)
type	Type of argument to be retrieved
arg_ptr	Pointer to list of arguments
va_alist	Name of parameter to called function (UNIX version only)
va_dcl	Declaration of **va_alist** (UNIX version only)
Returns	The current argument

Retrieves a value of *type* from the location given by *arg_ptr*. Note that **va_list** is defined as the type of **va_alist**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

va_end

void va_end(**va_list** *arg_ptr* **);**

Include	STDARG.H or VARARGS.H
arg_ptr	Pointer to list of arguments
va_alist	Name of parameter to called function (UNIX version only)
va_dcl	Declaration of **va_alist** (UNIX version only)
Returns	No return value

Resets *arg_ptr* to **NULL**. Note that **va_list** is defined as the type of **va_alist**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

va_start

void va_start(va_list *arg_ptr* **);** (UNIX version)

void va_start(va_list *arg_ptr*, *prev_param* **);** (ANSI version)

Include	STDARG.H or VARARGS.H
arg_ptr	Pointer to list of arguments
prev_param	Parameter preceding first optional argument (ANSI version only)
va_alist	Name of parameter to called function (UNIX version only)
va_dcl	Declaration of **va_alist** (UNIX version only)
Returns	No return value

Sets *arg_ptr* to beginning of list of optional arguments passed to the function used before **va_arg**. Note that **va_list** is defined as the type of **va_alist**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

vfprintf

int vfprintf(FILE ***stream*, **const char** ***format*, **va_list** *argptr* **);**

Include	STDARG.H or VARARGS.H, STDIO.H
stream	Pointer to target stream
format	Format control string
argptr	Pointer to list of arguments
Returns	The number of characters written if successful; a negative value if not

Formats and outputs data to *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

vprintf

int vprintf(const char **format*, **va_list** *argptr*);

Include	STDARG.H or VARARGS.H, STDIO.H
format	Format control string
argptr	Pointer to list of arguments
Returns	The number of characters written, not counting the terminating null character ('\0'), if successful; a negative value if not

Formats and outputs data to the standard output stream, **stdout**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

vsprintf

int vsprintf(char **buffer*, **const char** **format*, **va_list** *argptr*);

Include	STDARG.H or VARARGS.H, STDIO.H
buffer	Storage location for output
format	Format control string
argptr	Pointer to list of arguments
Returns	The number of characters written, not counting the terminating null character ('\0'), if successful; a negative value if not

Formats and outputs data to *buffer*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

wait

int wait(int **termstat* **);**

Include	PROCESS.H, ERRNO.H
termstat	Child process termination status and return code buffer
Returns	The child process identification if returning after normal child process termination; –1 if returning after abnormal child process termination: or –1 immediately otherwise
errno	**ECHILD, EINTR**

Suspends the calling process until any of the caller's immediate child processes terminate. If all of the caller's children have terminated before it calls the **wait** function, the function returns immediately.

☐ ANSI ☐ DOS ■ OS/2 ■ UNIX ■ XENIX

_wrapon

short _far _wrapon(short *option* **);**

Include	GRAPH.H
option	**_GWRAPOFF, _GWRAPON**
Returns	The previous value of *option*

Controls whether text output with the **_outtext** and **_outmem** functions wraps to a new line or is simply truncated when the text output reaches the edge of the current text window.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

write

int write(int *handle*, **void** ***buffer*, **unsigned int** *count*);

Include	IO.H, ERRNO.H
handle	Handle referring to open file
buffer	Data to be written
count	Number of bytes
Returns	The number of bytes actually written if successful; –1 if not
errno	**EBADF, ENOSPC**

Writes *count* bytes from *buffer* into the file associated with *handle*. The write operation begins at the current position in the file. If the file is open for appending, the operation begins at the current end of the file.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

yn

double yn(int *n*, **double** *x*);

Include	MATH.H, ERRNO.H
n	Integer order
x	Floating-point value
Returns	The result of a Bessel function of *x*; **–HUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order *n*).

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_ynl

long double _ynl(int *n*, **long double** *x* **);**

Include	MATH.H, ERRNO.H
n	Integer order
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*; **–_LHUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order *n*). Long double-precision floating-point version of **yn**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

y1

double y1(double *x* **);**

Include	MATH.H, ERRNO.H
x	Floating-point value
Returns	The result of a Bessel function of *x*; **–HUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order 1).

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_y1l

long double _y1l(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*; **–_LHUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order 1). Long double-precision floating-point version of **y1**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

y0

double y0(double *x* **);**

Include	MATH.H, ERRNO.H
x	Floating-point value
Returns	The result of a Bessel function of *x*; **–HUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order 0).

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_y0l

long double _y0l(long double *x* **);**

include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*; **–_LHUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order 0). Long double-precision floating-point version of **y0**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

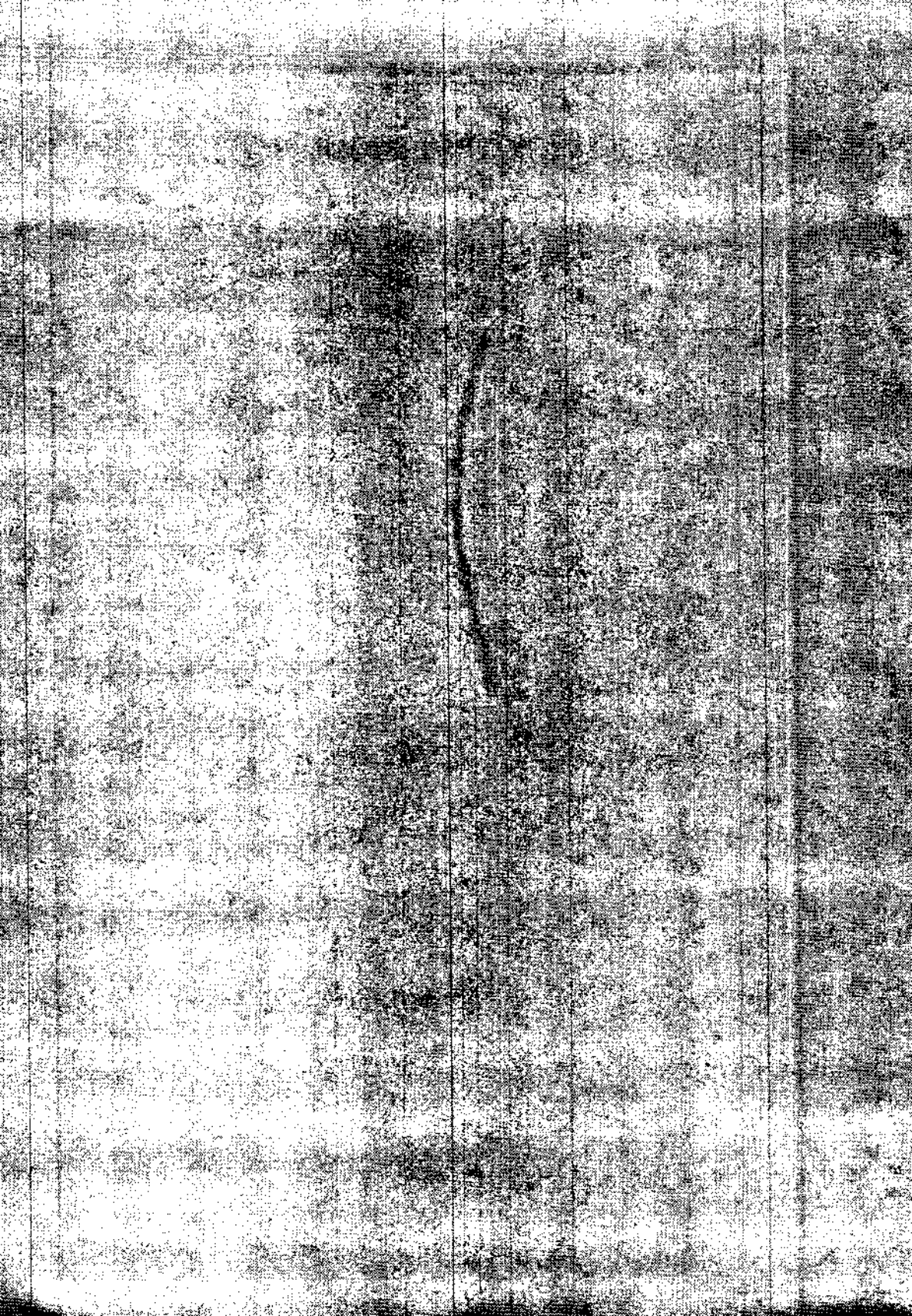
Appendixes

A printf/scanf Format Specifiers 369

B Compiler Limits and Numerical Ranges 373

C ASCII Codes 377

D Key Codes 379



Appendix A
printf/scanf Format Specifiers

The following gives the format syntax for **printf** and **scanf**.

printf Syntax

% [[*flags*]][[*width*]] [[.*precision*]][[{F|N|h|l|L}]]*type*

scanf Syntax

% [[*]][[*width*]] [[{F|N}]][[{h|l}]]*type*

Field	Description		printf	scanf
flags	Characters that justify output and control the printing of signs, blanks, decimal points, and octal and hexadecimal prefixes		Yes	No
	Code	Description		
	–	Left justifies	Yes	No
	+	Prefixes signed output with + or – Always printed with sign	Yes	No
	0	Adds leading zeros to reach minimum width	Yes	No
	blank (' ')	Prefixes zero or signed positive value with a blank	Yes	No
	#	1. With **o, x, X**: Prefixes nonzero output value with 0, 0x, or 0X 2. With **e, E, f**: Inserts decimal point 3. With **g, G**: Inserts decimal point and does not truncate trailing zeros	Yes	No
*	Suppresses assignment of the next field		No	Yes
width	Specifies minimum width in characters. For **printf**: If *width* is an asterisk (*), the width is determined by the next argument, an integer. This width argument precedes the argument being formatted.		Yes	Yes

Continued on next page

Field	Description	printf	scanf
precision	Specifies precision in number of digits and decimal places. If *precision* is an asterisk (*), the precision is determined by the next argument, an integer. This precision argument precedes the argument being formatted.	Yes	No
	1. With **d**, **i**, **u**, **o**, **x**, **X**: Specifies minimum number of digits. If number is less than *precision*, pads output value on the left with zeros. Does not truncate values larger than *precision*. 2. With **e**, E: Specifies number of digits after the decimal point and rounds the last printed digit. 3. With **f**: Specifies number of digits after the decimal point. 4. With **g**, **G**: Specifies the maximum number of significant digits. 5. With **c**: Has no effect. 6. With **s**: Specifies maximum number of characters to be printed.		
F	Explicitly indicates **far** value	Yes	Yes
N	Explicitly indicates **near** value	Yes	Yes
h	1. With **d**, **i**, **o**, **x**, **X**: Specifies **short int** 2. With **u**: Specifies **short unsigned int**	Yes	Yes
l	1. With **d**, **i**, **o**, **x**, **X**: Specifies **long int** 2. With **u**: Specifies **long unsigned int** 3. With **e**, **E**, **f**, **g**, **G**: Specifies **double**	Yes	Yes
L	With **e**, **E**, **f**, **g**, **G**: Specifies **long double**	Yes	Yes

Continued on next page

Field	Description		printf	scanf
type	Characters that justify output and control the printing of signs, blanks, decimal points, octal and hexadecimal prefixes		Yes	Yes
	Code	Meaning		
	c	Single character	Yes	Yes
	d	Signed decimal integer	Yes	Yes
	e, E	Exponential; case sets case of exponent key	Yes	Yes
	f	Floating-point value	Yes	Yes
	g, G	**e** or **f** format; case sets case of exponent key	Yes	Yes
	i	For **printf**: Signed decimal integer For **scanf**: Signed decimal, octal, or hexadecimal integer	Yes	Yes
	n	Number of bytes successfully written (**printf**) or read (**scanf**) placed in its corresponding argument, which is a pointer to an integer. No input or output is performed.	Yes	Yes
	o	Unsigned octal integer	Yes	Yes
	p	Pointer to **void**: prints address pointed to by the argument	Yes	Yes
	s	Null-terminated string	Yes	Yes
	u	Unsigned decimal integer	Yes	Yes
	x, X	For **printf**: Unsigned hexadecimal integer using "abcdef" or "ABCDEF" For **scanf**: Unsigned hexadecimal integer	Yes	Yes

Appendix B
Compiler Limits and Numerical Ranges

Compiler Limits

Limits Imposed by the C Compiler

Item	Limit
String literals	2,048 bytes, including the terminating null character ('\0')
Constants	Determined by its type; see on-line help
Identifiers	31 bytes (additional characters are discarded)
Declarations	15 levels of nesting for structure and union definitions
Macro definition	6K bytes, 255 formal arguments
Macro expansion	6K
Preprocessor arguments	3K (approximately)
if, **#ifdef**, and **#ifndef** directives	16 levels of nesting
Include files	10 levels of nesting
Initialization	30 levels of nesting

Program Limits at Run Time

Item	Limit
File size	$2^{32} - 1$ bytes (4 gigabytes)
Open files (streams)	20[a]
Command line (DOS)	128 characters (including program name)
Environment table (DOS)	32K
Command line and environment table (OS/2)	32K, combined

[a]The default number of open files and streams allowed can be modified. See _NFILE_ and **CRT0DAT** in on-line help.

Numerical Ranges

Data Ranges Defined in LIMITS.H

Manifest Constant	Description	Value
CHAR_MAX	Maximum **char** value	127[a]
CHAR_MIN	Minimum **char** value	–127[a]
CHAR_MAX	Maximum **char** value	255[a]
CHAR_MIN	Minimum **char** value	0[a]
SCHAR_MAX	Maximum signed **char** value	127
SCHAR_MIN	Minimum signed **char** value	–127
UCHAR_MAX	Maximum unsigned **char** value	255
CHAR_BIT	Number of bits in a **char**	8
USHRT_MAX	Maximum unsigned **short** value	65,535
SHRT_MAX	Maximum (signed) **short** value	32,767
SHRT_MIN	Minimum (signed) **short** value	–32,767
UINT_MAX	Maximum unsigned **int** value	65,535
ULONG_MAX	Maximum unsigned **long** value	4,294,967,295
INT_MAX	Maximum (signed) **int** value	32,767
INT_MIN	Minimum (signed) **int** value	–32,767
LONG_MAX	Maximum (signed) **long** value	2,147,483,647
LONG_MIN	Minimum (signed) **long** value	2,147,483,647

[a] These values are defined within an **#ifndef** block as follows:

```
ifndef _CHAR_UNSIGNED
#define CHAR_MAX    127
#define CHAR_MIN    (-127)
#else
#define CHAR_MAX    255
#define CHAR_MIN    0
#endif
```

Numerical Values Defined in FLOAT.H

Manifest Constant	Description	Value
DBL_DIG	Number of decimal digits of precision	15
DBL_EPSILON	Smallest value such that 1.0+**DBL_EPSILON** != 1.0	2.2204460492503131e–016
DBL_MANT_DIG	Number of bits in mantissa	53
DBL_MAX	Maximum value	1.7976931348623158e+308
DBL_MAX_10_EXP	Maximum decimal exponent	308
DBL_MAX_EXP	Maximum binary exponent	1,024
DBL_MIN	Minimum positive value	2.2250738585072014e–308
DBL_MIN_10_EXP	Minimum decimal exponent	307
DBL_MIN_EXP	Minimum binary exponent	–1,021
FLT_DIG	Number of decimal digits of precision	7
FLT_EPSILON	Smallest value such that 1.0+**FLT_EPSILON** != 1.0	1.192092896e–07F
FLT_MANT_DIG	Number of bits in mantissa	24
FLT_MAX	Maximum value	3.402823466e+38F
FLT_MAX_10_EX	Maximum decimal exponent	38
FLT_MAX_EXP	Maximum binary exponent	128
FLT_MIN	Minimum positive value	1.175494351e–38F
FLT_MIN_10_EXP	Minimum decimal exponent	-37
FLT_MIN_EXP	Minimum binary exponent	–125
LDBL_DIG	Number of decimal digits of precision	19
LDBL_EPSILON	Smallest value such that 1.0+**LDBL_EPSILON** != 1.0	5.42101086242752217O6e–020
LDBL_MANT_DIG	Number of bits in mantissa	64
LDBL_MAX	Maximum value	1.189731495357231765e+4932L
LDBL_MAX_10_EXP	Maximum decimal exponent	4,932
LDBL_MAX_EXP	Maximum binary exponent	16,384
LDBL_MIN	Minimum positive value	3.3621031431120935063e-4932L
LDBL_MIN_10_EXP	Minimum decimal exponent	-4,931
LDBL_MIN_EXP	Minimum binary exponent	–16,381

Appendix C
ASCII Codes

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01	☺	SOH
^B	2	02	☻	STX
^C	3	03	♥	ETX
^D	4	04	♦	EOT
^E	5	05	♣	ENQ
^F	6	06	♠	ACK
^G	7	07	•	BEL
^H	8	08	◘	BS
^I	9	09	○	HT
^J	10	0A	◙	LF
^K	11	0B	♂	VT
^L	12	0C	♀	FF
^M	13	0D	♪	CR
^N	14	0E	♫	SO
^O	15	0F	☼	SI
^P	16	10	►	DLE
^Q	17	11	◄	DC1
^R	18	12	↕	DC2
^S	19	13	‼	DC3
^T	20	14	¶	DC4
^U	21	15	§	NAK
^V	22	16	▬	SYN
^W	23	17	↨	ETB
^X	24	18	↑	CAN
^Y	25	19	↓	EM
^Z	26	1A	→	SUB
^[	27	1B	←	ESC
^\	28	1C	∟	FS
^]	29	1D	↔	GS
^^	30	1E	▲	RS
^_	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127†	7F	⌂

† ASCII code 127 has the code DEL. Under DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key combination.

Dec	Hex	Char
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

Dec	Hex	Char
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	░
177	B1	▒
178	B2	▓
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Dec	Hex	Char
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	█
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

Dec	Hex	Char
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	φ
238	EE	ε
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	ⁿ
253	FD	²
254	FE	■
255	FF	

Appendix D
Key Codes

Key	Scan Code		ASCII or Extended†			ASCII or Extended† with SHIFT			ASCII or Extended† with CTRL			ASCII or Extended† with ALT		
	Dec	Hex	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
ESC	1	01	27	1B		27	1B		27	1B				
1 !	2	02	49	31	1	33	21	!				120	78	NUL
2 @	3	03	50	32	2	64	40	@	3	03	NUL	121	79	NUL
3 #	4	04	51	33	3	35	23	#				122	7A	NUL
4 $	5	05	52	34	4	36	24	$				123	7B	NUL
5 %	6	06	53	35	5	37	25	%				124	7C	NUL
6 ^	7	07	54	36	6	94	5E	^	30	1E		125	7D	NUL
7 &	8	08	55	37	7	38	26	&				126	7E	NUL
8 *	9	09	56	38	8	42	2A	*				127	7F	NUL
9 (	10	0A	57	39	9	40	28	(				128	80	NUL
0)	11	0B	48	30	0	41	29	)				129	81	NUL
- _	12	0C	45	2D	-	95	5F	_	31	1F		130	82	NUL
= +	13	0D	61	3D	=	43	2B	+				131	83	NUL
BKSP	14	0E	8	08		8	08		127	7F				
TAB	15	0F	9	09		15	0F	NUL						
Q	16	10	113	71	q	81	51	Q	17	11		16	10	NUL
W	17	11	119	77	w	87	57	W	23	17		17	11	NUL
E	18	12	101	65	e	69	45	E	5	05		18	12	NUL
R	19	13	114	72	r	82	52	R	18	12		19	13	NUL
T	20	14	116	74	t	84	54	T	20	14		20	14	NUL
Y	21	15	121	79	y	89	59	Y	25	19		21	15	NUL
U	22	16	117	75	u	85	55	U	21	15		22	16	NUL
I	23	17	105	69	i	73	49	I	9	09		23	17	NUL
O	24	18	111	6F	o	79	4F	O	15	0F		24	18	NUL
P	25	19	112	70	p	80	50	P	16	10		25	19	NUL
[{	26	1A	91	5B	[	123	7B	{	27	1B				
] }	27	1B	93	5D	]	125	7D	}	29	1D				
ENTER	28	1C	13	0D	CR	13	0D	CR	10	0A	LF			
CTRL	29	1D												
A	30	1E	97	61	a	65	41	A	1	01		30	1E	NUL
S	31	1F	115	73	s	83	53	S	19	13		31	1F	NUL
D	32	20	100	64	d	68	44	D	4	04		32	20	NUL
F	33	21	102	66	f	70	46	F	6	06		33	21	NUL
G	34	22	103	67	g	71	47	G	7	07		34	22	NUL
H	35	23	104	68	h	72	48	H	8	08		35	23	NUL
J	36	24	106	6A	j	74	4A	J	10	0A		36	24	NUL
K	37	25	107	6B	k	75	4B	K	11	0B		37	25	NUL
L	38	26	108	6C	l	76	4C	L	12	0C		38	26	NUL
; :	39	27	59	3B	;	58	3A	:						
' "	40	28	39	27	'	34	22	"						
` ~	41	29	96	60	`	126	7E	~						

† Extended codes return NUL (ASCII 0) as the initial character. This is a signal that a second (extended) code is available in the keystroke buffer.

Key	Scan Code		ASCII or Extended†			ASCII or Extended† with SHIFT			ASCII or Extended† with CTRL			ASCII or Extended† with ALT		
	Dec	Hex	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
L SHIFT	42	2A												
|	43	2B	92	5C	\	124	7C	|	28	1C				
Z	44	2C	122	7A	z	90	5A	Z	26	1A		44	2C	NUL
X	45	2D	120	78	x	88	58	X	24	18		45	2D	NUL
C	46	2E	99	63	c	67	43	C	3	03		46	2E	NUL
V	47	2F	118	76	v	86	56	V	22	16		47	2F	NUL
B	48	30	98	62	b	66	42	B	2	02		48	30	NUL
N	49	31	110	6E	n	78	4E	N	14	0E		49	31	NUL
M	50	32	109	6D	m	77	4D	M	13	0D		50	32	NUL
, <	51	33	44	2C	,	60	3C	<						
. >	52	34	46	2E	.	62	3E	>						
/ ?	53	35	47	2F	/	63	3F	?						
R SHIFT	54	36												
* PRTSC	55	37	42	2A	*		INT 5§		16	10				
ALT	56	38												
SPACE	57	39	32	20	SPC	32	20	SPC	32	20	SPC	32	20	SPC
CAPS	58	3A												
F1	59	3B	59	3B	NUL	84	54	NUL	94	5E	NUL	104	68	NUL
F2	60	3C	60	3C	NUL	85	55	NUL	95	5F	NUL	105	69	NUL
F3	61	3D	61	3D	NUL	86	56	NUL	96	60	NUL	106	6A	NUL
F4	62	3E	62	3E	NUL	87	57	NUL	97	61	NUL	107	6B	NUL
F5	63	3F	63	3F	NUL	88	58	NUL	98	62	NUL	108	6C	NUL
F6	64	40	64	40	NUL	89	59	NUL	99	63	NUL	109	6D	NUL
F7	65	41	65	41	NUL	90	5A	NUL	100	64	NUL	110	6E	NUL
F8	66	42	66	46	NUL	91	5B	NUL	101	65	NUL	111	6F	NUL
F9	67	43	67	43	NUL	92	5C	NUL	102	66	NUL	112	70	NUL
F10	68	44	68	44	NUL	93	5D	NUL	103	67	NUL	113	71	NUL
NUM	69	45												
SCROLL	70	46												
HOME	71	47	71	47	NUL	55	37	7	119	77	NUL			
UP	72	48	72	48	NUL	56	38	8						
PGUP	73	49	73	49	NUL	57	39	9	132	84	NUL			
GREY -	74	4A	45	2D	-	45	2D	-						
LEFT	75	4B	75	4B	NUL	52	34	4	115	73	NUL			
CENTER	76	4C				53	35	5						
RIGHT	77	4D	77	4D	NUL	54	36	6	116	74	NUL			
GREY +	78	4E	43	2B	+	43	2B	+						
END	79	4F	79	4F	NUL	49	31	1	117	75	NUL			
DOWN	80	50	80	50	NUL	50	32	2						
PGDN	81	51	81	51	NUL	51	33	3	118	76	NUL			
INS	82	52	82	52	NUL	48	30	0						
DEL	83	53	83	53	NUL	46	2E	.						

† Extended codes return NUL (ASCII 0) as the initial character. This is a signal that a second (extended) code is available in the keystroke buffer.

§ Under DOS, SHIFT + PRTSC causes interrupt 5, which prints the screen unless an interrupt handler has been defined to replace the default interrupt 5 handler.

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

Microsoft®
Making it all make sense™

1089 Part No. 06515